

DRAFT SF 298

1. Report Date (dd-mm-yy) October 1995		2. Report Type		3. Dates covered (from... to)	
4. Title & subtitle Reengineering Technology Report Volume 1				5a. Contract or Grant #	
				5b. Program Element #	
6. Author(s)				5c. Project #	
				5d. Task #	
				5e. Work Unit #	
7. Performing Organization Name & Address				8. Performing Organization Report #	
9. Sponsoring/Monitoring Agency Name & Address Software Technology Support Center OO-ALC/TISE 7278 4th Street Hill AFB, UT 84056-5205				10. Monitor Acronym	
				11. Monitor Report #	
12. Distribution/Availability Statement Distribution Statement A: Approved for public release, distribution is unlimited.					
13. Supplementary Notes					
14. Abstract					
15. Subject Terms					
Security Classification of			19. Limitation of Abstract	20. # of Pages	21. Responsible Person (Name and Telephone #) Randy Wright (801) 777-9732
16. Report Unclass	17. Abstract Unclass	18. This Page Unclass			

STSC

**Reengineering
Technology Report**

Volume 1

October 1995

19970516 097

Document No: TRF-RE-9510-00.04

This report was prepared by:

Michael R. Olsem
Software Technology Support Center
Ogden ALC/TISE
7278 4th Street
Hill AFB, UT 84056-5205
Phone: (801) 775-5555, ext. 3057; DSN 775-5555, ext. 3057
FAX: (801) 777-8069, DSN 777-8069
Email: olsemm@software.hill.af.mil

Representations: The ideas and findings in this report should not be construed as an official Air Force position. It is published in the interest of scientific and technical information exchange.

References: The STSC makes every attempt to acknowledge the sources of information used, including copyrighted material. If, for any reason, a reference has been misquoted or a source used inappropriately, we would like it brought to our attention for correction.

Trademarks and Designations: Many of the designations and product titles used by manufacturers and sellers to distinguish their products are claimed as trademarks. The STSC has made every attempt to supply trademark information about manufacturers and their products mentioned in this report. The trademarks are the property of their respective owners.

This document is available through the STSC. To obtain a copy, please contact the STSC Customer Service Office directly at:

Software Technology Support Center
Attn: Customer Service
Ogden ALC/TISE
7278 4th Street
Hill AFB, UT 84056-5205

Voice: 801-775-5555, ext. 3024, DSN 775-5555, ext. 3024
FAX: 801-777-8069, DSN 458-8069

PREFACE

The purpose of this report is to increase contact, awareness, and understanding of software reengineering tools. Use of this report should be the first step in transferring effective software reengineering processes, methods, and tools into practical use. The targets of this report are organizations responsible for the development and maintenance of computer software. This report defines the ideas of software reengineering and identifies their value in improving software quality for embedded, MCCR, and MIS applications. It explains how the features of current reengineering tools can improve software development and maintenance. It includes information about specific products in the marketplace. The information is aimed at those who must make the decisions about acquiring advanced technology and prepare their organizations to use it effectively. Also, this report attempts to identify the future directions of the field to help in planning long-range strategies.

Because a reengineering effort involves large amounts of code and a significant percentage of a system (if not the entire system), it may be necessary to move to Ada. The official Air Force policy states "*Ada is required when more than one-third of the existing code is altered (excluding COTS) at any one time. Under one-third waiver not required. System managers are encouraged to move to Ada with any software or hardware upgrade.*" [Programming Languages Policy 5.c, 7 Aug 90] The STSC supports this policy and believes that converting to Ada during the reengineering process is the opportune time to comply with the Ada mandate.

Comments on this document are welcome and encouraged. Please address suggested changes and comments to:

Michael R. Olsem
Software Technology Support Center
Ogden ALC/TISE
7278 4th Street
Hill AFB, UT 84056-5205
Phone: (801) 775-5555, ext. 3057; DSN 775-5555, ext. 3057
FAX: (801) 777-8069, DSN 777-8069
Email: olsemm@software.hill.af.mil

This Page Intentionally Left Blank

TABLE OF CONTENTS

1. The Domain of Software Reengineering	1
1.1 What is Reengineering and How Can It Help Me?	1
1.2 The Growing Problems of Maintaining Software	2
1.3 Redevlopment vs. Reengineering.....	3
1.4 Continuous Improvement vs. Reengineering	3
1.5 Business Process Reengineering (BPR).....	3
1.6 A Taxonomy of Reengineering Terms, Tool Types, and Strategies ...	4
1.7 Tools vs. Services.....	6
1.8 Repositories.....	7
2. Reengineering Tools Database and Appendices	9
2.1 Introduction.....	9
2.2 Appendix A: Reengineering Tools List.....	9
2.3 Appendix B: Product Sheets	10
2.4 Appendix C: Product Critiques.....	10
2.5 Appendix D: Available Training and Conferences.....	10
2.6 Appendix E: Bibliography and Recommended Readings.....	10
2.7 Appendix F: Glossary	10
2.8 Appendix G: Software Technology Support Center Overview	10
3. Introduction to the Software Reengineering Assessment Handbook	11
3.1 Scope	11
3.2 Applicability	11
3.3 Motivation	12
3.4 Software Reengineering Assessment (SRA) Process	13
3.5 Content	14
4. A Reengineering Process Model	15
4.1 Introduction.....	15
4.2 Step 1: Define Organizational and Software Goals/Directions.....	16
4.3 Step 2: Form an Integrated Reengineering Team	16
4.4 Step 3: Define a CMM-Driven Development/Maint. Environment	17
4.5 Step 4: Select a Standard Set of Software Metrics	17
4.6 Step 5: Analyze the Legacy Software	18
4.7 Step 6: Define a Reengineering Implementation Process.....	19
4.8 Step 7: Develop/Update a Standard Testbed or Validation Suite	21
4.9 Step 8: Reengineering Tools Analysis	22
4.10 Step 9: Train, Train, Train.....	22
4.11 Other DoD Sources for Reengineering Process Work	23
4.12 Summary	23

Table of Contents

5. Pitfalls of Reengineering	25
5.1 The Politics of Reengineering	25
5.2 Technical Problems of Reengineering	25
5.3 Inherent Limits to Automated Reengineering	26
5.4 Legal Problems of Reengineering	26
5.5 Design Repository Problems	27
5.6 Your Organization's Resources	27
5.7 The Problems of Restructuring	28
5.8 Line-for-Line Source Code Translators	28
5.9 Reengineering Problems within the DoD	29
5.10 Reengineering Embedded or MCCR Software	30
5.11 Execution Time vs. Maintainability	30
5.12 Summary	31
6. Reengineering Projects and Services at the STSC	33
6.1 Introduction	33
6.2 SRAH	33
6.3 Reengineering Projects Survey	34
6.4 IEEE's <i>Reverse Engineering Newsletter</i>	35
6.5 The STSC <i>Reengineering Technology Report</i>	35
6.6 STC's Reengineering Track	35
6.7 Reengineering and BPR Tools Data Base	35
6.8 9-Step Reengineering Preparation Workshop	35
6.9 Introductory Reengineering Tutorial	35
6.10 COBOL Reengineering (COBRE)	36
6.12 Summary	36
7. Future Directions	37
7.1 Introduction	37
7.2 Repository Technology	37
7.3 Procedure-Oriented to Object-Oriented Translations	37
7.4 Single-Thread to Parallel Translations	39
7.5 Artificial Intelligence	40
7.6 Business Rules Extraction	40
7.7 Summary	41
Appendix A: Reengineering Products List	A-1
A.1 Reengineering Products List by Type	A-1
A.2 Reengineering Products List by Name (<i>see Volume 2</i>)	A-8
A.3 Reengineering Products List by Vendor (<i>see Volume 2</i>)	A-8
Appendix B: Reengineering Products Sheets (<i>see Volume 2</i>)	B-1
Appendix C: Reengineering Products Critiques (<i>see Volume 2</i>)	C-1
Appendix D: Available Training and Conferences	D-1
Appendix E: References and Recommended Readings	E-1
Appendix F: Glossary	F-1
Appendix G: Software Technology Support Center Overview	G-1

1.1 WHAT IS REENGINEERING AND HOW CAN IT HELP ME?

Reengineering is the bridge used by legacy software to migrate to an organization's new maintenance environment.

Traditionally, software engineering techniques have attempted to improve the process of new software development. These efforts have produced structured analysis/design, object-oriented analysis/design, Computer-Aided Software Engineering (CASE), etc. But what does an organization do with its legacy (i.e., existing) software created prior to the adoption of these wonderful new methodologies? Legacy software still needs to be maintained even though its quality, performance, reliability, and maintainability is deteriorating.

Reengineering tools can capture design information from otherwise indecipherable software (i.e. spaghetti code). These tools can supplement your legacy documentation to comply with DoD-STD-2167A, MIL-STD-498, or whatever documentation standard your organization uses. Unstructured software can be structured. And software/data can be ported to new languages, configurations, or platforms.

Software reengineering cannot exist in a vacuum. Reengineering must bridge to a defined target maintenance environment. This idea will be explored further in Section 4, but it's important to note that any reengineering project must fit within the framework of an organization's strategic/tactical plan.

Other reasons to reengineer include:

- Allow legacy software to quickly adapt to changing requirements
- Comply to new organizational standards (e.g., migrate legacy software to Ada)
- Upgrade to newer technologies/platforms/paradigms (e.g., object-oriented)
- Extend the software's life expectancy
- Identify candidates for reuse
- Improve software maintainability
 - Increase productivity per maintenance programmer
 - Reduce reliance on programmers who have specialized in a given software system
 - Reduce maintenance errors and costs

Volume 1 of this technology report discusses the current state of reengineering—its strategies, terminology, and future. Volume 2 lists all the reengineering tools and services that exist on the STSC data base at time of publication. We welcome your comments on this report and stand ready to assist any DoD organization looking into this technology. The contact information for the author (Michael R. Olsem) is listed in the Preface.

1.2 THE GROWING PROBLEMS OF MAINTAINING SOFTWARE

Software maintenance (as defined by ANSI/IEEE-STD-729-1983) is the modification of a software product after delivery to correct errors, improve performance (or other attributes), or adapt to new requirements. Let's look at some statistics:

Software errors can be very expensive. In a recent study, the top 10 most expensive software errors were maintenance errors. In fact, the top 3 most expensive software errors involved a single line of source code and cost their respective organizations \$1.6 billion, \$900 million, and \$245 million [McClure 90].

Software maintenance is very costly. A few statistics easily bears this out:

- Maintenance costs (including personnel and hardware/software usage fees) run as high as 50-80% of the software life cycle resources or \$30 billion a year in the United States alone [Moad 90]. Approximately 50% of maintenance costs is spent on just understanding what the software does.
- A 1991 Gartner Group study estimates that 90% of all software resources will be required by maintenance activities by 1995.
- The DoD spends approximately \$24 billion on software each year. Maintenance accounts for about 70% of this budget. To cite one recent example, the software development costs for the F-16 jet fighter were \$85 million. The projected software maintenance costs are \$250 million [Suydam 87].
- It is estimated that it will cost \$75 billion to fix the year 2000 problem in the USA alone [ComputerWorld, Sept. 1994].

Maintenance personnel are getting scarce. According to USAF estimates, 25% of this country's entire 18-to-25-year-old population will be required to maintain all our software systems by the year 2000 [Bush 88]. By the end of 1993, the DoD employed approximately 369,500 software personnel. By the year 2002, this force structure (and its budget) is to be reduced by 25% [Browning 93].

In addition, software maintenance typically face the following problems:

- Frequent failure rates
- Complex design (e.g. unstructured code, tightly coupled to hardware or other software, low cohesion, unknown development process, etc.)
- Unpredictable "ripple" effects
- Unreliable or missing documentation
- Obsolete hardware platforms
- Loss of experienced maintenance programmers or original developers
- Growing backlogs

All of the preceding problems are magnified when considered within an environment of shrinking budgets. But it is more economically feasible to address your organization's maintenance problems than it is to improve software development. Assume software maintenance accounts for 80% of a software system's lifecycle costs while new development accounts for 20%. If you double the efficiency of the development process (using CASE tools, etc.) then you've freed up 10% of your organization's software resources (half of the 20% formerly used for new development). But if you double the maintenance productivity, you free up 40% of your organization's software resources (half of the 80% formerly used for maintenance). Considering the DoD spends approximately \$16.8 Billion a year on maintenance (roughly 70% of its entire software budget of \$24 Billion) then we're talking about a \$6.7 Billion savings in the DoD alone (40% of \$16.8 Billion).

1.3 REDEVELOPMENT VS. REENGINEERING

If the preceding problems become severe enough (and the legacy software is important enough) then redevelopment is the traditional solution. But this is not always a good idea for the following reasons:

- Critical corporate knowledge is contained within the legacy software
Legacy software represents an enterprise model of your organization. If the software has been in use for a long time, then it has most likely survived by providing a critical service. In fact, some software engineers believe that most organizations have already built the majority of their useful software systems. The original users and developers may no longer be available to explain all the reasons behind the creation of, and subsequent modifications to, the software.
- Legacy software is a valuable asset
According to some estimates, software development runs about \$8-20 per LOC. Since the 1950's, over 100 billion LOC have been written—80% of which is COBOL. This represents an investment in COBOL code alone of between \$640 billion and \$1.6 trillion. This asset is second only to oil reserves or the global marketplace.
- Reusable, reengineered software costs much less than redeveloped code
Estimates of reusing reengineered software is about \$2-\$5 per LOC. As seen previously, newly developed code costs \$8-20 per LOC.

Of course, redevelopment cannot be ruled out. Valid reasons exist for completely replacing legacy software. But for the preceding reasons, software reengineering should be considered a viable alternative.

1.4 CONTINUOUS IMPROVEMENT VS. REENGINEERING

Some organizations have decided that software reengineering and a new maintenance environment is too radical a step to try. Instead, these organizations gradually improve their maintenance environment using better tools, processes, or people. In the world of quality improvement, this is termed continuous improvement. But recent studies indicate the return-on-investment (ROI) for software reengineering far outweighs the benefits accrued through continuous improvement. A well-run continuous improvement plan can expect no more than 15% savings of resources (personnel time and money). Most of the Baldrige Quality Award winners report only 5-12% savings due to continuous improvement. But software reengineering surveys are showing savings on the order of 150-200% (ref. STSC reengineering projects survey).

Again, this is not to say a continuous improvement plan should not be considered. The organization's goals should be carefully considered before embarking on either a reengineering project or a continuous improvement plan. If savings of 15% will satisfy management goals over the long term, then the less risky strategy is continuous improvement. But if your organization anticipates budget reductions (or personnel losses) in excess of the best that continuous improvement can achieve, then software reengineering should be considered a viable alternative.

1.5 BUSINESS PROCESS REENGINEERING (BPR)

A new business strategy now challenges software organizations. It is called business process reengineering (BPR). The reader should never confuse business process reengineering with software reengineering. But we have included this sub-section because software reengineering is often driven by the requirements of BPR.

“(Business Process) Reengineering is the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed.” —Michael Hammer and James Champy

1 The Domain of Software Reengineering

Traditionally, organizations are structured vertically. Business units may include manufacturing, marketing, administration, etc. But vertical structuring is often a barrier to quick adaptation to the demands of a rapidly changing world. When confronted with a new objective, client, strategy, etc., every business unit must be mobilized from the top down. Thus, administration gears up, marketing gears up, manufacturing gears up, etc. Business process reengineering says this is wasteful of time and resources. The organization should be cross-functional so that integrated work teams focus on critical processes. This re-organization reaches into management so that instead of a vice-president in charge of marketing or R&D, there may be a vice-president in charge of the manufacturing and delivery of product XYZ or service ABC.

So what is the role of software reengineering?

Legacy software was developed to support business functions within the traditionally vertical organization structure. Thus, organizations have software to support marketing, manufacturing, etc.

Software reengineering can capture the software design information. Using new tools and techniques, this design information can be broken up into functionally cohesive chunks. These chunks are then analyzed and regrouped around the newly identified key business process. This regrouping is termed re-aggregation.

Notice that in many ways, this sounds a lot like the process of translating process-oriented software to object-oriented software (ref. Section 7 of this report). Software should reflect a meta-model of the real-world. In the past, organizations attempted to force their software to conform to a structure that did not match the real-world as understood by the eventual users of the software. This often caused communications problems between software designers and users. Now, with object-oriented analysis and BPR, organizations are re-aligning their software (and organizational structure) so that they correspond to real-world objects (and processes).

1.6 A TAXONOMY OF REENGINEERING TERMS, TOOL TYPES, AND STRATEGIES

The debate over domain definitions within reengineering is far from being resolved. During 1992, several reengineering conferences brought up the issue of a standard reengineering taxonomy. This makes sense when you have listened to far too many presentations that begin with the presenter's definitions so that the audience will have a common frame of reference.

In September of 1992, the Joint Logistics Commanders (JLC) workshop on software reengineering set forth definitions that the STSC has subsequently adopted [JLC, 1992]. These definitions are largely based on ideas gleaned from STSC's 1992 Reengineering Tools Report, the Chikofsky and Cross article to IEEE [Chikofsky, 1990], the 3rd Annual Systems Reengineering Technology Workshop held at the Naval Surface Warfare Center in Maryland [NSWC, 1992], and the 3rd Reverse Engineering Forum held at Northeastern University [Chikofsky, 1992].

DEFINITIONS

Systems Engineering:

"The top level process of engineering a system to meet overall requirements."

Software Reengineering:

"The examination and alteration of an existing subject system to reconstitute it in a new form. This process encompasses a combination of sub-processes such as reverse engineering, restructuring, redocumentation, forward engineering, and retargeting."

Aside: The new form of the subject system could be structured code - from "spaghetti" code; design information in graphical form - from input code; or the translation of the source code from one language to another while preserving the system's functionality.

¹ The STSC maintains a category in the reengineering tools data base for tools that support this form of reengineering.

Reverse Engineering¹:

“The engineering process of understanding, analyzing, and abstracting the system to a new form at a higher abstraction level.”

Aside: This higher abstraction level is understood within the context of the software system's lifecycle. The classic waterfall development lifecycle calls for requirements/specifications followed by design, code, test, implement, and maintain. Thus, if we start with the code, reverse engineering will extract design information which is at a higher abstraction level in the lifecycle.

Forward Engineering¹:

“Forward engineering is the set of engineering activities that consume the products and artifacts derived from legacy software and new requirements to produce a new target system.”

Aside: Notice the difference between software engineering and forward engineering. A hot topic within software reengineering circles is whether we even need the term “forward engineering” since this implies the normal development lifecycle sequence of events. If this was the extent of forward engineering, then forward engineering and software engineering can be considered identical terms. But we define forward engineering as using the output of reengineering. This implies that some reengineering must have occurred prior to the forward engineering activity. For example, the most common forward engineering activity involves the generation of source code from design information which was captured by a previous reverse engineering activity.

Data Reengineering¹:

“Tools that perform all the reengineering functions associated with source code (reverse engineering, forward engineering, translation, redocumentation, restructuring/normalization, and retargeting) but act upon data files.”

Aside: As in source code reengineering, data reengineering must have a goal or target configuration. We call this a target meta model. For example, relational data bases (RDBs) are a desirable target meta model (currently, less than 5% of all data files are RDBs). Data reengineering tools help to translate flat files and hierarchical files to RDB's.

Each reengineering activity for source code has a corresponding activity in data reengineering. Reverse engineering of data captures the design information of data files. Restructuring data could normalize that data. Data translation can move data files from a flat file configuration to a relational data base. And retargeting data files assists the migration of data to new platforms.

Data name rationalization (DNR) is a special case of data reengineering. DNR tools enforce uniform naming conventions across all software systems. DoD studies predict a 100 to 1 reduction in data names if such names were standardized and controlled across all services [Connall 92]. As Lt. General Peter Kind remarked at STC '94 “standard data is as important as standard equipment”.

Redocumentation¹:

“The process of analyzing the system to produce support documentation in various forms including users manuals and reformatting the systems' source code listings.”

Aside: Sometimes, the output of reverse engineering is thought to be the same as redocumentation. After all, when reverse engineering captures the design information from the legacy source code, the resulting information usually includes data flow diagrams, control flow charts, etc. The difference between redocumentation and reverse engineering is that the redocumentation usually generates system documentation according to a standard. For example, there are redocumentation tools that create documentation for DoD-STD-2167A - a DoD software documentation standard.

A special case of redocumentation tools are reformatting tools. Otherwise known as “pretty printers”, reformatters make source code indentation, bolding, capitalization, etc. consistent thus making the source code more readable.

Restructuring¹:

“The engineering process of transforming the system from one representation form to another at the same relative abstraction level, while preserving the subject system’s external functional behavior.”

Aside: Recall that reverse engineering can extract design information from source code. Design is a higher level of abstraction than code (ref. most lifecycle charts). Restructuring code leaves the system at the same abstraction level (i.e. the coding level) but radically rearranges the source code to fit a new paradigm such as structured code or object-oriented code.

Retargeting¹:

“The engineering process of transforming and hosting or porting the existing system in a new configuration.”

Aside: The new configuration could be a new hardware platform, a new operating system, or a CASE platform. A good rule of thumb is that if more than 20-30% of the software must be changed during a retargeting project, then redeveloping the software specifically for the new platform might be a better strategy.

Source Code Translation¹:

“Transformation of source code from one language to another or from one version of a language to another version of the same language (e.g., going from COBOL-74 to COBOL-85).”

Business Process Reengineering¹:

“The fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed.” —Hammer and Champy

Aside: Tools that support BPR include process modelers (and simulators) that allow organizations to run what-if scenarios on their key business processes. Other BPR tools enable an organization to set goals and gather information about current or projected processes.

1.7 TOOLS VS. SERVICES

This report lists both tools and services. When is it appropriate to use a reengineering service instead of COTS tools? There are basically two scenarios that may make the service more practical:

- There are no COTS tools that fit
If your organization’s primary source code language is not mainstream, or your reengineering needs are in some way unique (such as a proprietary data base configuration), then turning to a service may be your only reengineering option. Several tool vendors also provide reengineering services and may be able to modify their tools to handle your unique requirements.
- Sometimes it’s more economical to use a service
Reengineering tools are rarely cheap. If the software needing reengineering is not large, then the cost tradeoff of using a service vs. buying a tool (which you may never use again after your software has been reengineered) might easily favor using a service.

If you choose a service, do not expect your reengineering responsibilities to be over. The core process (as outlined in the Reengineering Process Model of Section 4) must still be performed by your organization. More importantly, you must work closely with any service to ensure correct reengineering. War stories abound where a service returns reengineered software that is no where near what was expected. As a software organization, you probably recall users that ordered new software and then complained that the resulting system is not what they requested. That’s why prototyping was developed. In the role of customer to the reengineering service, don’t make the same mistake.

1.8 REPOSITORIES

Identified as a key component in the DoD's *Software Technology Strategy* [DoD 91], a repository should be central to your organization's reengineering efforts (see Figure 1-1). Reverse engineering tools require a place to store extracted design parameters. But the repository is much more:

- Requirements-based maintenance

The repository will change the very nature of maintenance and development by focusing on the change **requirements**, not on the technical problems of change and implementation. This allows accountants, engineers, scientists, etc. to directly modify a system without an intervening layer of software specialists (application programmers and analysts). Changes can be made and tested at the requirements level. When ready to execute, a code generator creates the source code in the language deemed most efficient for the given application and platform. This frees an organization from reliance on a given language's quirks and highly-trained/highly-paid personnel.

- Center for reusable software components

Traditional reuse repositories are based on code fragments that have been encapsulated and thoroughly validated. But programmers seem reluctant to use these repositories due to 3 basic reasons:

- Distrust of the reusable code fragment (the "not invented here" syndrome)
- The time required to understand what the reusable code fragment does
- The reusable code fragment includes a lot more functionality than is needed

But reusable design repositories don't have these problems. Since the reusable design fragment is graphically represented, the programmer can immediately determine its functionality and easily identify/remove any excess functionality. Once the new software is fully developed from reusable design fragments, a code generator (i.e., forward engineering tool) can then create the executable source code.

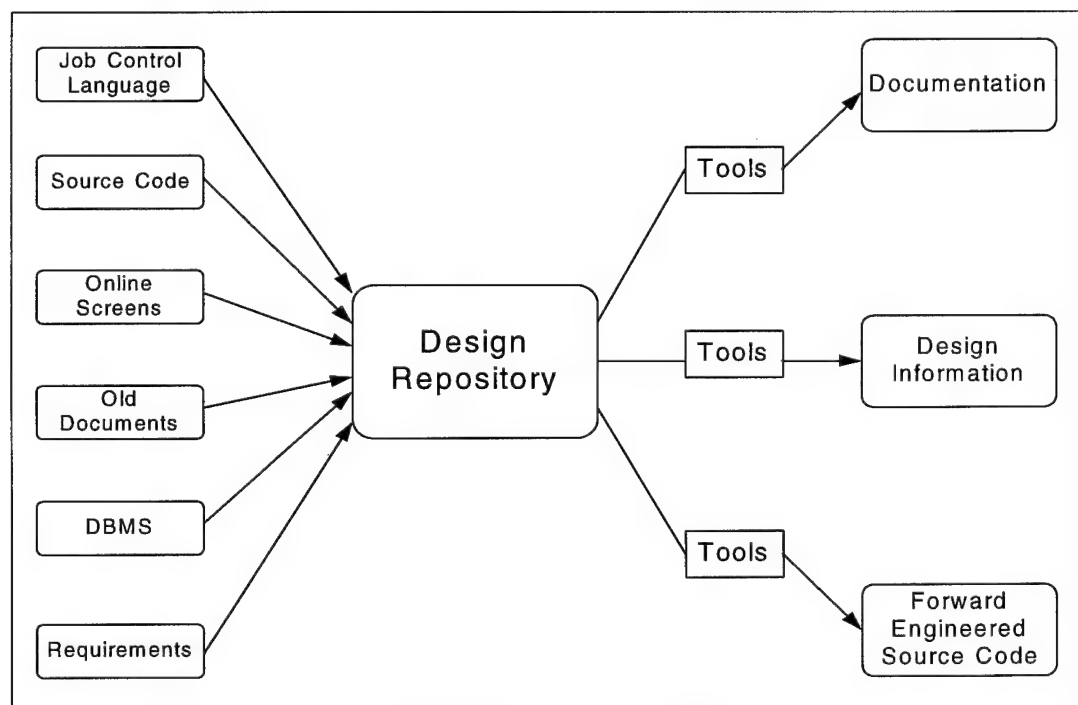


Figure 1-1. Repository-Based Reengineering/Maintenance

- Central development/maintenance site for the entire organization
 - Allows different developers to coordinate their work on the same software
 - Enables an architectural approach to software development/maintenance. (Contains the goals, requirements, specifications, validation suite, report designs, screen designs, data models, process models, business rules, data structures, strategies, hardware components, data dictionary, data flow diagrams, program structures, and the organization's meta-models for BPR.)
 - Integrates tools for validation, data base management, configuration management, etc. by providing a common interface for different languages, DBMS's, I/O interfaces, and methodologies from a variety of vendors. Thus, it must embrace an open architecture.
 - Provides for a methodology-guided exploration and maintenance [Sayani 91]. Given a formal maintenance environment, each change or fix would follow a step by step process (i.e. methodology). For example, from a given trouble report: root causes are hypothesized, errors localized, potential ripple effects studied, solution efforts estimated, and corrections validated.

It should be pointed out that not all the functionality described above is available from all repositories. The I-CASE initiative is attempting to address many of these benefits.

2

REENGINEERING TOOLS DATABASE AND APPENDICES

2.1 INTRODUCTION

Due to the current size of the tools list, we have included the entire tools list (Appendix A), product sheets (Appendix B), and some product critiques (Appendix C) in Volume 2 of this report. Most readers are not interested in searching through the entire printed list to find those tools that match their reengineering requirements. The STSC will create a customized tools list based on your reengineering requirements—for free. Just call us with your requirements and we will search our entire data base for tools that fit your reengineering needs. **Thus, Volume 2 is available upon request only.** If you are a vendor, please request Volume 2 and validate your tool information. Contact us immediately if you disagree with any of the data, if your tool is missing, or the tool's functionality has expanded beyond our current description.

We welcome any and all suggestions regarding these appendices. If you have information that should be included in this report (e.g., a reengineering conference), let us know. Your input will continue to shape and improve this report.

2.2 APPENDIX A: REENGINEERING TOOLS LIST

The alphabetized lists of reengineering tools can be found in Volume 2 of this report. Appendix A contains all reengineering tools known at time of publication. The tools list is sorted (and printed) by three different criteria:

- Appendix A.1: Lists the tools by reengineering type (i.e., sub-domain)
- Appendix A.2: Lists the tools in alphabetical order by tool name
- Appendix A.3: Lists the tools in alphabetical order by vendor name

Each line within the tool lists contains the following tool information:

- The tool's name
- Compatible hardware platforms and associated operating systems
- Vendor name and phone number
- Computer languages and data bases supported by the tool
- The tool's reengineering sub-domain types. Currently, these include (see Section 1 for definitions):
 - Reverse Engineering
 - Forward Engineering
 - Source Code Translator
 - Redocumentation
 - Restructure
 - Retargeting
 - Data Reengineering
 - Business Process Reengineering

The reader will note that additional sub-domains are listed next to some tools in Appendix A. These sub-domains are used and discussed in other Technology Reports published by the STSC—primarily the *Software Test Technologies Report*. The *Software Test Technologies Report* also includes a discussion and full list of the STSC data base on software analysis tools.

2.3 APPENDIX B: PRODUCT SHEETS

We attempt to contact every vendor for more detailed tool information using the STSC product sheet. These product sheets are listed in Volume 2 of this report and are available upon request.

The additional information provided by the product sheets include:

- Cost, discounts, site licenses
- A more detailed description
- Development/maintenance methodologies supported by the tool
- Frequency of new releases
- How tool documentation is updated by the vendor
- Minimal required hardware
- Minimal required software
- Misc. services such as a vendor BBS, hotline, newsletter, training, tool customization, GUI support, Windows support, etc.

We try to uncover any hidden hardware/software requirements for each tool. For example, if a tool requires an upgrade to the hardware platform, such as a math co-processor, then this would be included in the product sheet for that tool.

2.4 APPENDIX C: PRODUCT CRITIQUES

Every vendor normally maintains a list of satisfied customers as “character” references. In addition, the STSC actively solicits experienced users’ opinions of these reengineering tools. We are interested in perceived strengths and weaknesses of the tool and advice/warnings for potential tool buyers. To be fair, we have included a space for vendor comment/rebuttal of the users’ remarks. Some sample product critiques appear in Volume 2 of this report and are available upon request.

At the request of potential tool buyers, we will send a more detailed questionnaire to the experienced users who filled out product critiques. All such questionnaires will be held in confidence at the request of the tool user.

2.5 APPENDIX D: AVAILABLE TRAINING AND CONFERENCES

Appendix D is a list of conferences and training workshops in the field of software reengineering or directly relating to this field.

2.6 APPENDIX E: BIBLIOGRAPHY AND RECOMMENDED READINGS

Appendix E contains a list of articles and books regarding the software reengineering field. These represent some of the key readings which the STSC used when shaping our approach to this domain.

2.7 APPENDIX F: GLOSSARY

Terms and acronyms used throughout this paper have been collected in Appendix F for easy reference.

2.8 APPENDIX G: SOFTWARE TECHNOLOGY SUPPORT CENTER OVERVIEW

Appendix G provides an introduction to the Software Technology Support Center, including its mission, strategy, products, and services.

3 INTRODUCTION TO THE SOFTWARE REENGINEERING ASSESSMENT HANDBOOK

3.1 SCOPE

The *Software Reengineering Assessment Handbook (SRAH)* provides information and guidance to individuals responsible for DoD software, specifically in maintaining existing software, implementing software reuse, and incorporating Commercial Off The Shelf (COTS)/ Non-Developmental Items (NDI) into new or existing systems. It focuses on providing practical assistance to the analyst and/or engineer responsible for performing the analysis of existing software for potential reengineering. However, the results of the process described in the SRAH will be used by DoD decision makers as a basis for making informed decisions about whether to maintain, reengineer, or retire existing software. Portions of this process may not be fully applicable to every type of system and are tailorable to each user organization's needs.

3.2 APPLICABILITY

The SRAH applies to the analysis and planning of reengineering efforts for all DoD software. Use of this handbook should be consistent with the provisions of applicable directives. This handbook also applies to other Government and commercial/industry software, that is, the same principles and processes apply. It provides a defined Software Reengineering Assessment (SRA) process for conducting an effective technical, economic, and management assessment of existing software to determine whether to maintain, reengineer, or retire that software. The technical and economic assessment sub-processes are intended for execution by individuals having a thorough understanding of the existing system, software engineering/reengineering, and software economics, or access to someone who has this knowledge. The management decision sub-process is intended for execution by a decision maker familiar with these areas, but not necessarily expert in any of them.

The overall SRA process can be performed at the system level, the Computer Software Configuration Item (CSCI) level for embedded/tactical systems, or the computer program level for Automated Information Systems (AIS). Its applicability ranges from small to very large systems from a single maintainer to DoD executives responsible for software. The SRA process can be applied at the maintenance organization level, the program office level, or at the systems command level.

The SRA process is applicable to various types of project life cycle approaches including waterfall, incremental, periodic, and spiral. It will be applied more frequently for incremental, periodic, and spiral reengineering, than for waterfall, but on a smaller scale.

The SRA process is designed to be tailored to meet the needs of certain types/classes of systems. For example, when tactical (hard real-time, embedded, or weapon-related systems versus AIS) systems are considered, the reengineering decision maker (e.g., Program Manager) may not be choosing between several systems for the best to reengineer. Rather, he/she is faced with a go/no-go choice on reengineering, or is reengineering for other than pure economic reasons and only needs to choose the best available strategy. Thus, some of the handbook processes are not applicable. Furthermore, systems engineering trade-off analysis may precede the SRA to address open technical issues. Each user should tailor the SRA process to fit their organization's needs.

3.3 MOTIVATION

Existing software is a valuable DoD asset which must be used to the greatest degree possible. Consequently, there is an immediate need for DoD guidance for conducting technical, economic, and management assessment of reengineering techniques to determine when they are cost beneficial. The *Software Reengineering Executive Strategy and Master Plan* produced at the JLC-JPCG-CRM Santa Barbara I Reengineering Workshop states:

The state of DoD software systems is approaching a national crisis. Currently 30% of the software budget is being spent on development, while the other 70% is spent on maintenance. Woefully, 50% of the maintenance resources are expended on understanding system requirements and the design and specifications of existing systems. This catastrophe has drastically affected our warfighting capability and drained our resources. Resources currently devoted to continued maintenance could be better utilized on reengineering or new development with greater return on investment. This alarming trend will be exacerbated unless there is a devoted effort to change the entire process of the system acquisition from cradle to grave. Utilizing systems software reengineering and reuse as a paradigm for capitalizing on our investment in existing and future systems shows great potential for resolving a major portion of the problem ... The vision of this Master Plan is to preserve, extend, and leverage DoD's past and present investments in systems through reengineering for the future. Bold leadership is required to achieve the following goals for the year 2000:

- For existing systems – Reengineer based on return on investment assessment.
- For new systems – Develop with reengineering and/or reuse structures.
- For technology – Enable working at high levels of aggregation (e.g., configurations) and abstraction (e.g., requirements and specifications).
- For infrastructure – Foster consistent policies, standards, procedures, education, tools, incentives, and business practices that integrate reengineering into the systems engineering process.

Technical reasons to reengineer existing systems include mission requirements, major enhancements, modifications for new applications, rehosting to a new computer, translation to a new language, etc., which eventually comes down to an economics decision. This Handbook provides the required DoD guidance for conducting technical, economic, and management assessments to determine when software reengineering techniques are cost effective.

3.4 SOFTWARE REENGINEERING ASSESSMENT (SRA) PROCESS

The factors leading to the decision to reengineer are complex. The goal of the SRAH is to present those factors in a straightforward manner. Decisions regarding reengineering are critical since they involve the allocation of an organization's resources: *money, time, and personnel*.

Figure 3-1 presents an overview of the SRA Process. Subsequent SRAH chapters discuss the individual parts of the process in greater detail. There are multiple entry and exit points in the process. Candidate software (e.g., computer programs) enters the process based on a number of factors listed in SRAH (Section 4.2 on page 4-3) including those candidates perceived as being a problem. Candidates specifically directed to be reengineered will not need to be screened. Similarly, candidates that were previously identified through some other process for reengineering can enter the SRA process at the beginning of the economic process—the technical process is optional. However, the user may want to subject such candidates to the entire process to determine if a different strategy is more appropriate and/or to validate the process itself.

As previously described, the SRA process consists of three distinct components: *technical, economic, and management*.

A. Reengineering Technical Assessment (RTA) Process

1. Assess the organization's level of preparation to reengineer (SRAH Section 4.2).
2. Identify candidate software (computer programs A, B, C, D, and E, for example) (SRAH Section 4.3).
3. Screen the list of candidates to determine their viability for reengineering (programs A, C, and E, for example) (SRAH Section 4.4).

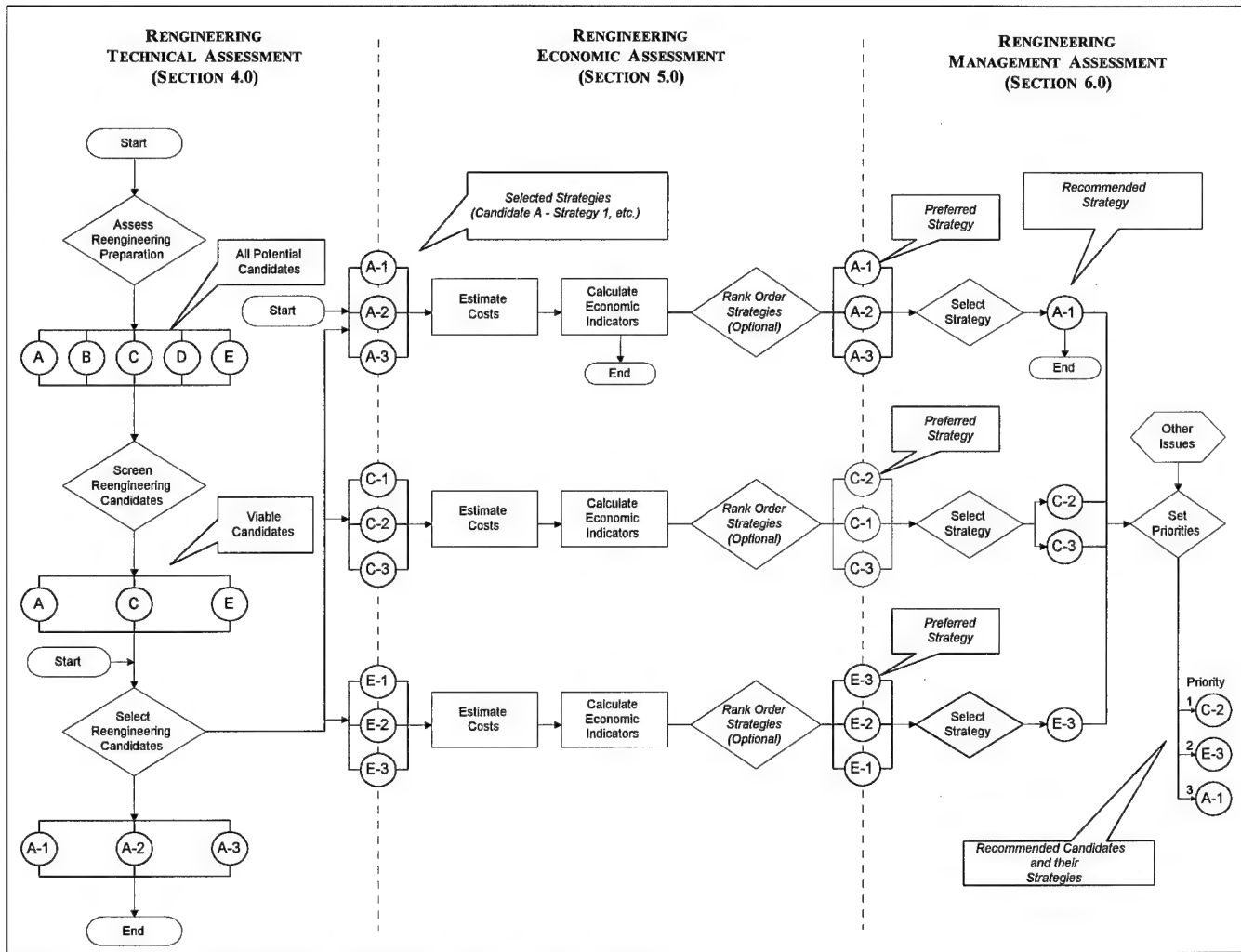


Figure 3-1. Software Reengineering Assessment (SRA) Process Overview

4. Complete the RTA Questions, Table 4-1, Page 4-14, for each candidate. Identify the reengineering strategies (A-1, A-2, A-3, C-1, C-2, C-3, E-1, E-2, and E-3, for example) based on the scores from the questionnaire (SRAH Section 4.5).
5. Consider other strategies (SRAH Section 4.6).
6. If evaluating multiple candidates, enter the RTA Questionnaire results into the Detailed Assessment Results (DAR) worksheet in appendix C, and answer the Maintenance Environment Question Set (SRAH Section 4.14, Page 4-28).
7. Document the technical assessment results.

B. Reengineering Economic Assessment (REA) Process

8. Estimate the cost of each strategy identified by the RTA Process. Strategy 1 (Status Quo) is always considered (SRAH Sections 5-3 through 5-6).
9. Calculate the economic indicators for each strategy (SRAH Sections 5-7 and 5-8).
10. Rank order the strategies using appropriate economic indicators (optional) (SRAH Section 5.9).
11. Document the economic assessment results (SRAH Section 5.10).

C. Reengineering Management Decision (RMD) Process

12. Prepare the Management Report according to SRAH section 6.2 (no form or worksheet is included). Select the recommended strategy for each candidate being evaluated. Review and confirm that the strategy meets all technical, economic, mission, user, customer, and organizational objectives. Select a different strategy if necessary. Using an appropriate economic and/or technical indicator, rank order the candidates being evaluated with their corresponding recommended strategies. Identify other considerations related to each candidate and/or groups of candidates that may influence the relative priorities. This may include organizational requirements and capabilities, user or customer requirements and funding, staff and time constraints.
13. Select the reengineering projects, and establish project priorities (SRAH Section 6.3).
14. Implement and document the management decision results. Include the basis for the priority assigned to each candidate. Integrate this documentation with the documentation from the RTA and REA (SRAH Section 6.4).

3.5 CONTENT

Volume I contains the technical, economic, and management decision sections and related Appendices A through D. Volume II, Appendices E through K, contains information, guidance, and examples for using specific models to estimate reengineering efforts. The examples were based on a case study which left much room for individual interpretation. Consequently, each model developer assumed slightly different attributes of the case study, which reengineering strategies were more beneficial/economical, and which approach should be taken for a particular strategy. The results of the various models should not be compared with each other because the assumptions differed and the models are based on different project databases. The purpose of providing these appendices is to show that the models do support estimating reengineering efforts. In some cases, the model developers provided significant insight and innovation based on their experiences with actual reengineering projects to date. This information adds significant credibility to the proposition that software reengineering can be very economical.

4

A REENGINEERING PROCESS MODEL

4.1 INTRODUCTION

Many of the USAF organizations we advise are far too eager to rush out immediately and buy a reengineering tool. This, we believe, is analogous to shopping for groceries without a list of projected meals. Not only do you rarely return with all you need, but you usually end up purchasing items you don't need at all.

Based on studies of software reengineering projects, the STSC has defined a 9-Step Reengineering Process Model (see Figure 4-1). These 9 steps should be used for planning any software reengineering project. These steps should be accomplished in the order they are listed but can overlap:

- Step 1: Define Organizational and Software Goals/Directions
- Step 2: Form an Integrated Reengineering Team
- Step 3: Define a CMM-Driven Development/Maintenance Environment
- Step 4: Select a Standard Set of Software Metrics
- Step 5: Analyze the Legacy Software
- Step 6: Define a Reengineering Implementation Process
- Step 7: Develop/Update a Standard Testbed or Validation Suite
- Step 8: Reengineering Tools Analysis
- Step 9: Train, Train, Train...

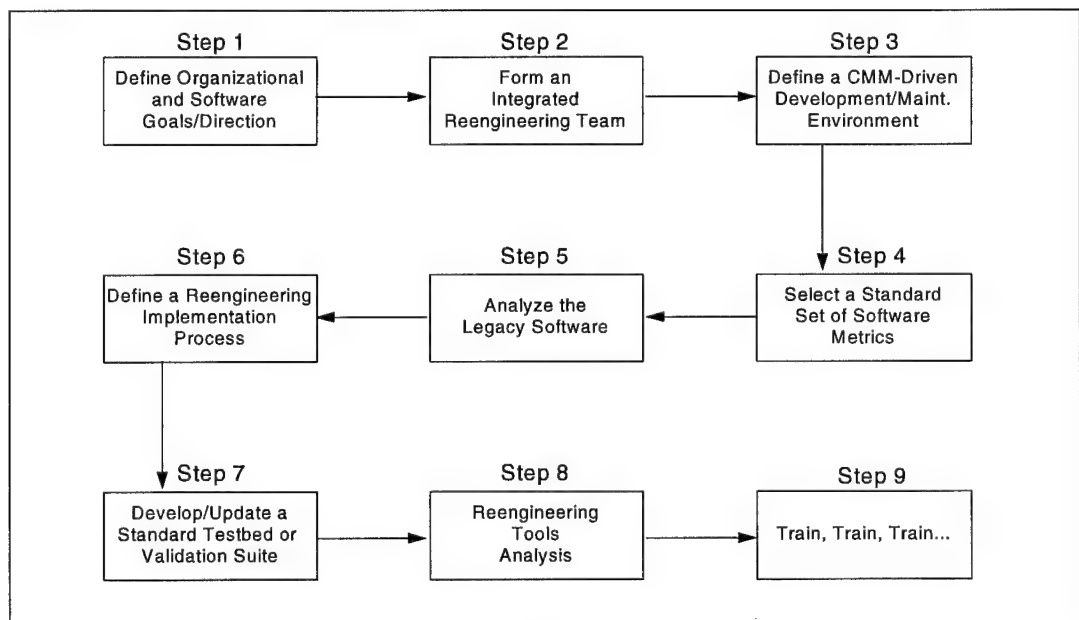


Figure 4-1. Reengineering Process Model

4.2 STEP 1: DEFINE ORGANIZATIONAL AND SOFTWARE GOALS/ DIRECTIONS

Your organization's strategic/tactical plans should drive any reengineering projects so that your reengineering goals agree with your organizational goals. Anything less and your reengineering project (tools and software) will probably become shelfware or at most, an interesting footnote. For example, translating source code to Ada should not be undertaken due to an external mandate. Translation to Ada should be undertaken due to well-defined goals such as easier maintainability, reuse, enhanced user communications, testability, etc.

The legacy software (to be reengineered) should also fit within your organization's long term goals. Reengineering is not inexpensive. Resources (tools, time, and money) spent to reengineer software can only be recaptured if the software is expected to have a strategic role for at least 5 years.

One method to define organizational goals is to have an SEI Capability Maturity Model (CMM) analysis performed. Such an analysis helps to identify process problems that should be addressed by modifications to your current development/maintenance environment. Tailor your reengineering project to migrate your legacy software into this new environment.

4.3 STEP 2: FORM AN INTEGRATED REENGINEERING TEAM

Create a reengineering team composed of those programmers, users, and managers that are knowledgeable and open-minded enough to evaluate whether technical change can solve their maintenance problems. Choose carefully. Team members should be technically competent, credible, patient, possess good social skills, and know the organization's culture well enough to function successfully.

Be sure the user community is well-represented. Besides their expertise, you need their support and buy-in. In addition, users help in the validation of the post-reengineered software.

The tasks of the reengineering team include [Seymour 92]:

- Begin/maintain documentation of the reengineering project/process and lessons learned (reference page 53 of MIL-STD-498 for documentation standards as applied to software reengineering projects)
- Understand the current environment and business needs
- Establish goals, strategies, and action plans
- Provide cost justification
- Test new tools
- Purchase tools
- Provide internal marketing, consulting, and service
- Train others
- Continue to research and evolve technology
- Provide vendor liaison - partnerships to advance technology
- Improve their own processes using client satisfaction surveys
- From the lessons learned during one or more pilot projects, develop a transition plan to institutionalize your reengineering process and apply it to larger software applications throughout the organization

This last point is important. Too often following a successful pilot project, no thought has been given regarding scaling issues (i.e., application of the reengineering tools and techniques to much larger software applications) or how to move the remaining legacy software to the new maintenance environment.

4.4 STEP 3: DEFINE A CMM-DRIVEN DEVELOPMENT/MAINTENANCE ENVIRONMENT

Input to this creation process should come from users, programmers, and managers. Consult the programmers working on the current development/maintenance process. Even if the current maintenance process is informal, it should have some merit. Change always has a better chance at success when it fits within the organization's cultural norms. But the users may want faster modification turn-around. And the managers are frustrated with their inability to make the software change as quickly as the environment demands.

The creation of a development/maintenance process is clearly beyond the scope of this report. It is much better covered by SEI's CMM. But be aware that this step is time-consuming and must be done prior to reengineering. If your organization already has a well-defined, well-regulated development/maintenance process, then clearly this step is satisfied.

4.5 STEP 4: SELECT A STANDARD SET OF SOFTWARE METRICS

"When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science." —Lord Kelvin.

"And everything else that we have talked about, whether it be technology, whether it be new tools such as I-CASE, or whether it be process improvement, whatever it might be in the final analysis, they mean nothing if we can't measure and document the level of improvement, and in fact you will not be able to sell to management those benefits unless you can demonstrate those savings." —Lloyd K. Mosemann (closing address to STC '93)

You must know where you are in order to use a map to get to where you want to go. You must be able to show management what they got in return for their money and support. Metrics prove to the skeptics that reengineering was worth all the pain you caused them. Because change is painful and the extent of change caused by reengineering can be extensive.

But beyond the political concerns, metrics can help you find out what is wrong and how to fix it. Think of software as you would a doctor's patient. A doctor gathers as much data about the state of the patient's health as he/she can to try to determine the root cause of a patient's problem prior to prescribing treatment. Similarly for software, data helps one determine and correct the root cause of a problem, not just correct the symptoms. Metrics and software analysis tools can gather this information for you. And just as your family doctor keeps a record of your health over the years, you should maintain an historical metrics record on your software for its entire lifecycle. You (and your doctor) need a baseline to determine when something is wrong in order to react as quickly as possible.

There are some basic metrics which have become defacto standards for measuring the health of software. The most common are McCabe's cyclomatic and essential complexity metrics. Cyclomatic complexity is the number of paths through the software, and thus, a measure of the modularity of the software. Obviously, the smaller the cyclomatic complexity, the easier the code is to maintain. Essential complexity is the number of deviations from good structured design contained within the software. Obviously, the closer to zero for essential complexity, the more structured the code. These metrics can be used to determine which software should be reengineering first. But they can also be used on an ongoing basis to determine the effect of maintenance on your software.

Metrics should be non-threatening to the people collecting them. Metrics should only be used for process improvement and the continued health of your software. Don't use metrics for personnel appraisals. Otherwise, your personnel will start reporting what they think you want to hear. Quality improvement requires honest, accurate, and consistent data from the people gathering the data. Also, make sure the people see that the data is being used, or they will stop providing it.

The metrics you choose for your organization should directly reflect your organization's goals as defined in Step 1 of this preparation process. Thus, metrics are used to define "success." If your goal was cheaper maintenance or faster maintenance, then metrics will measure your progress to that goal. But metrics will also help you to understand some of the trade-offs you may face to reach those goals. For example, easier maintenance may come at the expense of slower execution time. Your

reengineering planning needs to include thresholds which the users or programmers or management consider as minimal requirements. Such thresholds could include response time, the number of maintenance personnel required, the turn-around time for a modification request, the quality of the software (as measured in failure rate or mean time between fixes - MTBF), etc. Don't choose metrics that cannot be directly related to organizational or project goals. Collecting arbitrary metrics is a waste of everyone's time. A good way to determine appropriate metrics for your organization is to use Victor Basili's "Goal, Question, Metric Paradigm" [Basili 89].

1. Identify your goals (organizational and project)
2. Determine what questions need to be answered to achieve those goals
3. Determine what metrics you can collect to answer those questions

CrossTalk (Special Edition, 1993) contains a good article on software metrics by Capt. Mark Kanko (USAF) entitled "Of Gas Gauges and Software Metrics." In this article Capt. Kanko discusses the qualities of a good metric. These include simplicity, validity, robustness, prescriptiveness, and analyzability.

The STSC is teaching and consulting on the official USAF metrics policy (as put forth by Darlene Druyen - Deputy Secretary of the Air Force). For more information contact the STSC.

4.6 STEP 5: ANALYZE THE LEGACY SOFTWARE

Each candidate software may have different goals and correspondingly different reengineering strategies. Some software may have sufficient documentation but your maintenance process requires capturing the design information to a repository. Other software may have missing or inaccurate documentation requiring redocumentation and reverse engineering to a repository. Thus, it's important to inventory your legacy software and determine appropriate reengineering strategies (if any). Determine what software is critical to your organization and its customers. Perform a risk analysis for the candidate software. What would be the impact on your organization if a given piece of software were to fail? Maybe nobody would care. If you're feeling brave (or reckless), stop executing the software and see whether anybody notices. This is a sure-fire way to determine whether the software is being used and by whom.

The first section of JLC's SRAH (ref. Section 3) helps an organization to inventory their legacy software and match software candidates to appropriate reengineering strategies. The next section of the SRAH then calculates the cost to reengineer each candidate software. The final section of the SRAH suggests ways to use the resulting information (matched strategies and costs) to prioritize the reengineering projects.

Catalogue any unique aspects for eventual reengineering. For example, is the software comprised of several programs linked by Job Control Language (JCL)? Does it use any embedded DBMS or Assembler macro calls? Are other languages embedded or called? What about online screens? Are there any implied "dynamic" (i.e., execution time) requirements such as response time constraints, available memory, etc.? Moreover, this information should be captured within the design repository. When picking your repository or reengineering tools, ask the vendor whether their tool(s) can handle all your critical design information.

Gather whatever requirements information you can find. Currently, reengineering tools cannot derive requirements information from source code. But you need requirements data to effectively maintain your software systems. Talk to the original developers (if available), maintainers, users, management, and anyone else who's worked with the software from the time it was developed. Ask the vendor whether their repository can store requirements information.

There are numerous tools to help you gather analysis data on your legacy software. These tools analyze the complexity of your software, check the degree of structuredness, evaluate the impact of a given modification, and identify standards violations (e.g., non-initialized variables, dead code, etc.). This data will help during the validation phase (Step 7) and supply objective data to support your selection of the most appropriate reengineering strategy (Step 6) applicable to the candidate software. Program analysis tools do not modify source code. Running program analysis tools before and after

reengineering provides a means of measuring the effectiveness of the reengineering. These tools are described in much more detail in the STSC's Software Test Technologies Report [Test 94].

4.7 STEP 6: DEFINE A REENGINEERING IMPLEMENTATION PROCESS

Based on the preceding steps, you must now plan how to implement your reengineering process. Step by step, decide what needs to be done, taking into account the legacy software characteristics (starting point), your new maintenance process (finish point), and all intervening steps such as validation, metrics, tool integration, training, etc. Set up a viable schedule with milestones to report back to management.

One cautionary note: it is very tempting to include requirements modifications during the reengineering process. **DON'T!!!** Software requirement modifications will create enormous problems with functional validation and greatly complicate the whole project. Wait until the reengineering project is complete and stable (i.e., functionally validated) before implementing modifications. However, we realize that a large reengineering project may go on for months or even years. How does an organization maintain (i.e., allow modifications) a critical software system that is being reengineered. There are two basic strategies:

- Using good configuration management techniques, collect all change requirements and implement all of them once the reengineered software has been fully validated.
- Using an incremental approach, reengineer a portion of the software, validate it, modify it per the new requirements, then repeat this process until the entire system is reengineered.

There are three fundamental strategies for reengineering which can be summed up as evolutionary vs. revolutionary:

1. Systems Reengineering (Revolutionary)

An entire system is reengineered (Figure 4-2). Systems reengineering can be used for one-time reengineering projects where you need to solve an immediate problem for a particular application. The advantage to this strategy is that the system is brought into the newly defined maintenance environment all at once. A disadvantage is the high amount of risk associated with systems reengineering. Following reengineering, the entire system must now operate flawlessly. But can you guarantee that? Is the functionality intact? Were any bugs introduced during the reengineering (remember, there is a good deal of human intervention with any reengineering project). If you're confident you can control these potential problems, and the system is sufficiently small enough, then systems reengineering will allow you to quickly switch over from the old system to the reengineered system. Otherwise, perhaps one of the following strategies would be better.

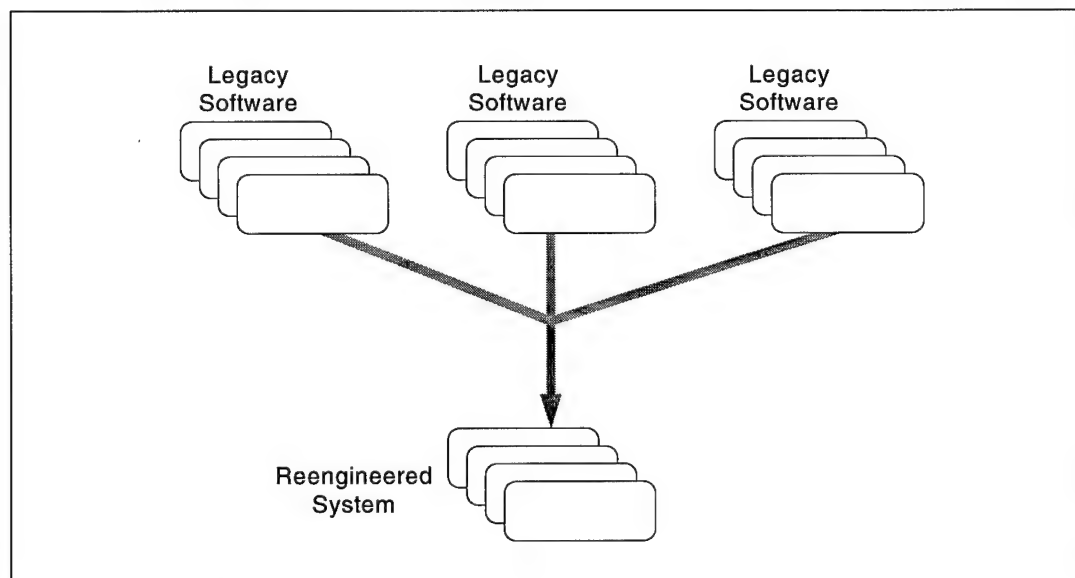


Figure 4-2. "System" Reengineering Process

2. Incremental Reengineering

Parts of the software are reengineered then re-integrated with the overall system (Figure 4-3). This approach creates versions of the software that must be managed using configuration control techniques. When a software modification is required, begin by first reengineering only those parts of the software that will be affected by the modification. Overall risk is reduced since only clearly defined parts of the software are changed. Should a problem occur, it's origin can be traced to those portions changed. The disadvantage lies in the number of interim versions of the software generated until all the software is eventually reengineered.

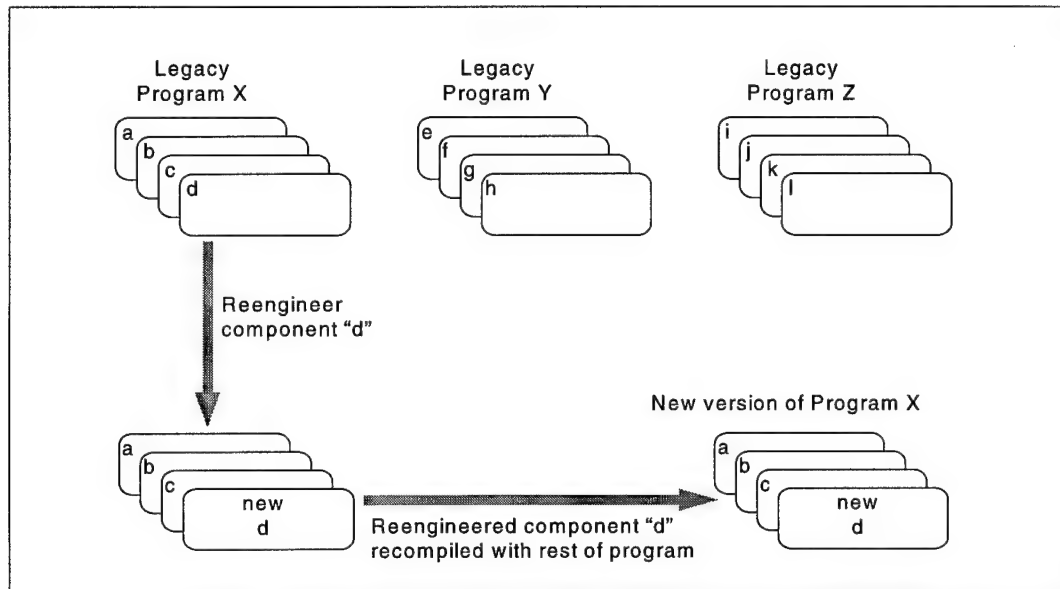


Figure 4-3. "Incremental" Reengineering Process

3. Partial Reengineering

Functionally cohesive modules of the software are reengineered as needed but **not** re-integrated with the rest of the software. When a software modification is required, first reengineer only those parts of the software affected by the modification (e.g., sub-system or sub-routine). But unlike incremental reengineering, the reengineered software is now separately maintained within the new software maintenance environment (Figure 4-4). For example, the reengineered module may now reside on a new hardware platform, CASE tool, or repository. The advantage to this strategy is the inherent advantage of modular design: if a problem occurs after the implementation of the reengineered software, just re-implement the old module. The disadvantage lies in managing the interface between the reengineered module and the rest of the software and whether the interface introduces any response time degradation that is unacceptable (particularly for real time software).

If reengineering is viewed as the path to long-term software maintenance improvement for your organization, then we recommend incremental or partial reengineering. But if you need a short-term solution to an immediate problem for a particular application, then systems reengineering might be the better strategy.

As with any new technology, reengineering should be inserted cautiously within an organization. Choose a software system (or module) that is relatively small and isolated (little or no impact on other software components and data files). Pilot projects enable an organization to test newly acquired tools and newly defined reengineering strategies. Lessons learned from pilot projects should then be applied to larger, more complex software reengineering projects. Thus, plan how to transition the tools, strategies, and lessons learned from the pilot project to all the organization's legacy software requiring reengineering.

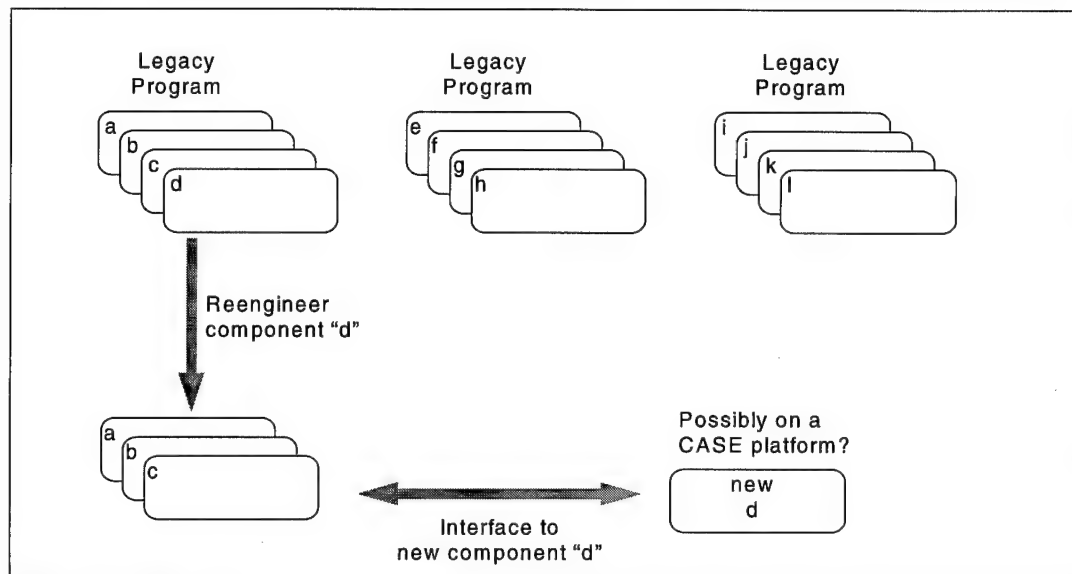


Figure 4-4. "Partial" Reengineering Process

4.8 STEP 7: DEVELOP/UPDATE A STANDARD TESTBED OR VALIDATION SUITE

This absolutely critical step of the reengineering process is often overlooked. How else do you prove the reengineered software is functionally equivalent to the legacy software? Whether you reformatted, restructured, translated, or generated code from captured design data, you need to validate the functionality of the resulting software. Critical software, in particular, need to demonstrate unchanged functionality after reengineering. This is a key reason why it is extremely unwise to propose any functional changes until after the reengineered software is validated. So check your current validation suite for the candidate software. Discover whether it's current and complete. If not, take the time to create/update a complete testbed for the candidate software.

Remember, the validation suite is also the minimum acceptance criteria for your users. We recommend that if your reengineering project uses multiple reengineering strategies (e.g., translate source code then restructure) that the software be validated after each strategy is completed.

As mentioned in the previous step, we strongly recommend against introducing any functional changes during a reengineering project. But if circumstances force such changes, then we recommend an incremental, spiral approach to reengineering projects which include functional enhancements. This is the reason why this step follows Step 6, so that the validation plan can match the chosen strategy for reengineering implementation. Essentially the incremental, spiral approach involves:

- Reengineer a modular piece of the software
- Validate the software that was reengineered
- Implement the functional enhancement(s)
- Supplement the validation test suite to allow for the enhancements
- Validate the enhancements
- Move on to the next modular piece of software to be reengineered(i.e., return to the top of this list)

Traceability is also important. Design functionality must be clearly associated with its corresponding source code. This is your only link between the new, repository-based design representation and the old source code. When looking at reengineering tools, ask the vendors about traceability.

For a more in-depth discussion of testing strategies, we strongly recommend you read the STSC report: *Software Test Technologies Report* [Test 94].

4.9 STEP 8: REENGINEERING TOOLS ANALYSIS

All the preceding steps should be accomplished prior to selecting any reengineering tool(s). “Buying software tools the way most software development organizations do is like going to the grocery store without a list—when you get home, not only do you not have everything you need, you have a lot of stuff you don’t need” [Meredith 91].

Find the reengineering tools that will accomplish your maintenance and reengineering goals. They should also fit the needs of your candidate software as defined previously (Step 6). Be aware that there may not be a COTS reengineering tool that satisfies all of your requirements. Your source code language may not have a sufficiently large user base to warrant any vendor creating a reengineering tool. Or there may exist reengineering tools for your candidate source code language but they won’t capture design information associated with JCL, online screen formats, DBMS calls, macros, etc. You may then need to decide whether the discrepancies between the tool’s functionality and the legacy software’s characteristics are sufficiently small to still make the tool usable.

You may have to “massage” your legacy software so that your reengineering tools can process it. This may take several forms:

- Most reengineering tools run on workstations or PC’s. Thus, if your software resides on the mainframe, there is the issue of downsizing or porting your legacy software from your mainframe.
- If the tool will only process COBOL 85 and your legacy software was written in COBOL 74 and COBOL 68, use a translator to upgrade to COBOL 85 code first. Another strategy would be to translate the code into a standard language first. For example, translate a platform-dependent assembler into Fortran before additional reengineering.
- If your legacy software includes embedded macros, DBMS calls, other language calls, etc., and your reengineering tool cannot process anything outside the primary source code language, then stub out that code.

A possible alternative to modifying your legacy source code to fit the reengineering tool, is to alter the reengineering tool to fit your legacy source code. Several vendors have accessible grammars that allow tool customization.

Tool integration is also an important consideration. As discussed above, no single reengineering tool will match all your requirements. But where one tool falls short, another may pick up the slack. So check whether the reengineering tool can pass data to other reengineering tools, your chosen repository, the hardware platform/OS, and the target maintenance process/tools.

And finally, the issue of scalability must be considered. A reengineering tool that works well for a small pilot project, may have serious problems when applied to larger software components. These problems include significantly slower execution time, insufficient hardware memory, inability to graphically represent overly complex designs, etc. Ask the vendor for a demonstration of their tool(s) on software components that correspond in size to the software you wish to reengineer.

The STSC maintains a database of all known commercial and government-owned reengineering tools and services (see Appendix A of this report). If you call the STSC, we will provide you a customized list of tools that meet as many of your requirements as possible. In addition, the STSC maintains a database of tool critiques about these tools written by objective users of these tools (see Appendix C of this report).

4.10 STEP 9: TRAIN, TRAIN, TRAIN...

Training seems to always be the last item budgeted. Rarely is an organization’s skills regularly updated. Training must be considered an ongoing task. Training is key to the success of any reengineering project and is important during each of the preceding 8 steps.

Government organizations cannot assume contractors will supply fully trained personnel and thus not budget for any additional training. Technology and processes are far too dynamic and require update training with an accompanying budget. Government organizations should expect contractors to supply “constantly trained” personnel, not “fully trained” personnel. However, if the government is

going to expect the contractor to continually (or regularly) have their people trained on the latest technology, then this needs to be spelled out in the contract. Otherwise, you will probably have to settle for the skills and knowledge that was present at the beginning of the contract, and no more.

Your reengineering team must be trained to understand several key concepts:

- How to manage technological change (see Section 5.1)
- The basics of reengineering
- How to use the selected reengineering tools
- How to use the target development/maintenance process

Because of the training and time investments in your reengineering team, you should seriously consider making the team permanent. Reengineering is an ongoing process. Any moderate to large software organization will have more than enough legacy software to keep such a team busy for the next several years at least. Which brings us to the last issue of training. By training the reengineering team, they will be able to train other key personnel involved in the reengineering project.

4.11 OTHER DoD SOURCES FOR REENGINEERING PROCESS WORK

In broad strokes, we have outlined a core process for planning a reengineering project. There are other reengineering planning processes currently available. The STSC reengineering data base includes several commercial planning processes. In addition, the government offers the following reengineering planning processes:

- DFAS is working on a reengineering process for each of the reengineering strategies. For example, if you decide to redocument, DFAS has a step by step process to redocument and fulfill any and all DoD requirements. There is also a process for software analysis, reverse engineering, forward engineering, and data name rationalization.
- The Army published a paper on reengineering process and decision strategy in 1991 entitled *Software Redesign Decision Making* [Army 1991]. Their basic premise is that three areas need to be consulted before deciding to reengineer: business, users, and technical considerations. In addition, they ask their readers to consider three basic questions to be answered by each group:
 - When should software be reengineered?
 - What software should be reengineered?
 - How should the software be reengineered?
- The Software Engineering Institute (SEI at Carnegie-Mellon University) plans to publish a "Best Practices" handbook for software reengineering. The STSC can help you contact the appropriate personnel within the SEI.
- The National Security Agency (NSA) has published the *Software Reengineering Guidebook*. The STSC can help you contact the appropriate personnel within the NSA.

The STSC also recommends Carma McClure's book *The Three R's of Software Automation: Reengineering Repositories Reusability* [McClure 90] and Patricia L. Seymour's tutorial on *Reengineering in the Real World* [Seymour 92]. Besides various reengineering strategies, Ms. Seymour discusses reengineering objectives, problems, plans, people, benefits, issues, and timing.

4.12 SUMMARY

Successful software reengineering requires careful planning. This level of planning also helps justify to management the amount of resources required. Smaller pilot projects are a good way of proving reengineering concepts and their cost savings. But whatever political strategy you employ, don't attempt reengineering without each of the steps outlined above, in the appropriate order.

This Page Intentionally Left Blank

5.1 THE POLITICS OF REENGINEERING

Change is always difficult and usually resisted. As agents of change, you walk a tightrope between the status quo and the future of software engineering. You must be sensitive to the politics and culture of your organization. Studies show that change stands a much better chance of success if it is phrased within the context of the organization's culture and values. This culture will probably not be written down anywhere so you must decipher it yourself or bring in people familiar with it into your reengineering team.

We recommend two courses for people acting as reengineering change agents. The first of these is SEI's class on *Managing Technological Change* at Carnegie-Mellon University. SEI can be reached at (412) 268-7700. This course teaches all aspects of how to introduce new ideas, methods, and tools into your software organization. This course is also available to government organizations, through the STSC. The point of contact at the STSC for this course is Mike Sturgeon at (801) 775-5555 extension 3069.

The second course comes from Technology Innovations and is called *Re-engineering in the Real World*. This course is much more reengineering-oriented and is taught by people who have managed reengineering projects. They can be reached at (510) 820-9448. As part of the course, a series of surveys help determine your reengineering needs, your consulting skills, level of client satisfaction, etc.

Both of these courses emphasize the need for infrastructure support to support a major change. Infrastructure support includes change agents, powerful sponsors, the reengineering team, and middle management. Without careful preparation and cultivation of these people, you will probably fail. Your career and reputation is on the line so do your homework.

5.2 TECHNICAL PROBLEMS OF REENGINEERING

Most reengineering tools work only on workstation and PC platforms. This may be due to the trend of client-server, distributed systems processing or it may be due to the versatility and power of mid-sized workstations. Whatever the cause, mainframe-based organizations interested in reengineering face downsizing/downloading of their software to run on these platforms. Thus, an interface issue is introduced both for input to the tool and returning its output to the production platform.

Reengineering tools still have a serious problem handling software written in multiple languages or with embedded macros, DBMS calls, etc. A software system is more than just the dominant source code language. Let's look at a typical MIS application. The dominant language is usually COBOL and there are plenty of COBOL reengineering tools available. But what about all of your DBMS calls. Will the tool handle these or must they be stubbed out? Do you have any embedded calls to Fortran or Assembler routines? Is the system online and if so, will the tool handle your I/O screen calls? What about the design logic contained in the JCL? This, too, must be captured and represented. If you're a DoD organization, will you have to generate Ada code to conform to DoD standards?

Usually a platform that allows an integration of tools is the best choice to handle this variety of input. This means the tools can exchange software meta-data. Remember, the reengineering tools will need to interact with metrics tools, validation tools, a repository, documentation tools, etc. As was mentioned in step 8 of the STSC reengineering planning process, the DoD has created the I-CASE (Integrated Computer-Assisted Software Engineering) project to allow suites of tools to exchange

5 Pitfalls of Reengineering

software meta-data. I-CASE was defined to allow an open architecture that encompasses development tools, maintenance tools, and reengineering tools. The STSC strongly recommends your organization look into I-CASE when planning your reengineering project.

The technology and tools that fit your needs may simply not exist. If your legacy software is not written in a source code language with a sufficient user base, no vendor will deem it economically feasible to create a tool for that language. The same can be said regarding your proprietary DBMS or data file system. So you are left with one of three choices:

- Manually reengineer or redevelop your systems
- Develop your own proprietary tools
- Contract with a software reengineering service to reengineer your legacy software for you

A cautionary note regarding this last option. Never turn your software systems over to a service company and assume the reengineering will be done to your complete satisfaction unless you stay intimately involved. Stay involved every step of the way and keep the application's users involved as well.

And finally, beware the problem of scalability. A small pilot project may successfully reengineer an organization's software but when applied to larger software applications (on the order of a million lines of code or more), some reengineering tools (or repository) start having problems. These problems include significantly slower tool response times (on the order of hours or days), insufficient memory, downloading congestion, and difficulty in re-integrating different software modules. Even the graphics will have problems displaying excessive design information. Our advice is to have the vendor(s) demonstrate their tool(s) using software of a magnitude similar to the software you need to reengineer.

5.3 INHERENT LIMITS TO AUTOMATED REENGINEERING

Currently we can't reverse engineer from source code to requirements. Capturing business rules is not the same as capturing requirements (see Section 7.6). Yet, the ideal maintenance process is based on requirements.

There are some implied requirements that can't be recovered from source code. For instance, if the system is embedded/real time, what response time constraints are required? Will your newly restructured, translated, or code-generated system run fast enough to meet user expectations and system demands. An F-16 pilot takes small comfort in knowing his arsenal is controlled by more maintainable software which is slower in responding to external threats.

Another set of implied requirements is that of the platform/operating system interface to your legacy software. Programmers take for granted the power and limitations of the platform/OS that run their systems. Thus, they add software routines to fill the gaps due to limitations of the platform/OS. Programmers also take for granted the strengths of their platform/OS and omit tests, routines, etc. that may be required by the target source code or new platform/OS. For example, different platform/OS's have different byte lengths, word lengths, precision, registers, etc. Memory requirements will differ. So make sure these requirements are accounted for.

5.4 LEGAL PROBLEMS OF REENGINEERING

If your legacy software uses COTS or third-party (i.e., contracted) software either by calling routines or as an integral part of the system, then be careful of some potential legal issues when reengineering.

Perhaps you want to enhance the COTS tool. One judge has ruled that such modification is possible if the change is necessary for the system to be executed. Another judge has ruled that such modification is legal if it makes the software more usable for the purposes for which it was acquired [Samuelson 90].

Perhaps you want to capture a COTS tool design into your repository because it is integral to your legacy systems. This may fall in the category of developing similar software at the design level representation. This is probably legal as long as you don't develop a competing software product.

Be aware that reverse engineering any source code that does not belong to you can carry some stiff penalties. "Copyright law has become the main battleground of the legal debate over whether reengineering technology can be used to reverse-engineer other firms' programs" [Samuelson 90]. Taking apart a competitor's product to see how it ticks has long been used as a strategy by many manufacturing firms. But, is this strategy legal in the case of software? The basic legal point is whether you made a copy of the software in the course of analyzing it. If so, then you may have infringed on the developer's copyright. Recent court rulings have been mixed on this point. The current trend seems to be that disassembly is legal as long as any resulting products are not too similar in design or look-and-feel.

Other reengineering court cases you may want to consult include:

- Stac v. Microsoft
- Vault v. Quaid
- Sega v. Accolade
- MAI v. Peak
- Nintendo v. Galoob
- Computer Associates v. Altai

These and other court cases are defining the scope of permissible reverse engineering. If there's any question regarding your reengineering project, consult a lawyer, give them the above references, and avoid actions that may give rise to lawsuits.

5.5 DESIGN REPOSITORY PROBLEMS

Design repositories are key to reengineering. They are the midpoint between reverse engineering and forward engineering. They should be key to any maintenance process that utilizes requirements-based modifications. Unfortunately, there aren't that many independent design repositories commercially available. Although many CASE tools have built-in repositories, they tend to be built around a pre-defined development/maintenance methodology. If this methodology matches your organization's target maintenance environment then you have a ready-made repository. If not, then you must find a design/requirements repository that suits your methodology or is at least neutral to any given methodology (such as the I-CASE environment).

CASE environments usually focus on software development from scratch. Software reengineering focuses on legacy software. Thus, CASE environments may have many auxiliary tools that are not applicable to the maintenance or reengineering of software. If you are strictly a maintenance organization, these auxiliary tools and processes may not only be unnecessary, but may actually get in the way as the CASE environment attempts to force its internal software engineering development process on you.

Many reengineering tools do not populate a repository. Restructuring tools simply restructure source code with the code as the output. The same can be said of redocumenters, reformatters, etc. Even reverse engineering tools (i.e., tools that capture design logic) do not necessarily send the design logic to a design repository. They may, instead, create graphical data flow diagrams, structure charts, control flow diagrams, etc. These graphical representations can be displayed on a screen or on paper, but not necessarily from a repository.

I-CASE attempts to provide a common interface between development, maintenance, and reengineering tools. If you decide to put together a non-I-CASE suite of software tools, then you need to look at the interface format used by each tool. These include PCTE, IGES, CDIF, etc. Make sure your tools can exchange meta-data with the repository using one of these common interface formats.

5.6 YOUR ORGANIZATION'S RESOURCES

Reengineering is not a cure-all for all the critical problems of maintaining legacy software. If nothing else, we hope you come away from this report with this idea:

Reengineering is the bridge used by legacy software to migrate to an organization's new maintenance environment.

There is no silver bullet. Only careful planning, resource commitment, and follow-through will save your software organization from the out-of-control expenses of future maintenance. You must ask yourself this question: Does your organization have the patience and short term resources to support a reengineering effort? Remember the steps outlined in Section 4 for planning a reengineering project. They included: evaluating business needs, forming a reengineering team, creating a development/maintenance process, defining metrics, creating a testbed, analyzing the legacy software, creating a tailored reengineering process, selecting the tools, and training. This is a considerable investment in time, budget, and personnel. Does your organization have the resources to initiate and support this level of effort?

5.7 THE PROBLEMS OF RESTRUCTURING

There are several problems inherent in restructuring source code. Chief among these is the maintenance of the output. Your software maintenance shop probably has one or more programmers whose job is to maintain the software you want to restructure. The original system may be difficult to maintain but at least you have that in-house expertise with their sometimes-intuitive understanding of the system. These are normally your organization's "heroes" who work long hours when a maintenance change causes bizarre side-effects (i.e., bugs) because the system is unstructured and poorly documented.

But after you restructure this source code, the resulting software will quite likely be incomprehensible to your software maintenance experts. The act of restructuring duplicates routines, moves routines, and alters routines—whatever it takes to make the source code conform to the restructuring tool's internal model of structured software design. The restructured source code may be more maintainable in the long run, but your in-house expertise will probably understand the new code about as well as a new maintenance programmer. In fact, several restructuring projects immediately reassigned responsibility for maintenance of the resulting software to programmers unfamiliar with the original system.

That is not to say restructuring is bad. In many cases, restructuring may be required as an interim step prior to reverse engineering. If the original source code is in really bad shape (i.e., totally incomprehensible) then restructuring, followed by the graphical display of the design information (i.e. reverse engineering), will produce a much more understandable design than attempting to analyze the graphical design representation of unstructured code (what with control flow and data flow process arrows going every which way).

5.8 LINE-FOR-LINE SOURCE CODE TRANSLATORS

Line-for-line source code translators create source code in a different language (than that used by the legacy system) by creating a new line of code for each line of legacy code. This process completely bypasses any design repository.

The problem with line-for-line translation is that it does not take advantage of the semantic constructs inherent in the new language. For example, when going from COBOL to Ada, line-for-line translation will not take advantage of Ada's object-oriented constructs. Some have derisively called such translations "Adabol".

Assuming the technology and tools exist (for both your legacy software's language and the new language), a better approach for source code translation would be to:

1. Capture the design information (via reverse engineering) to a design repository
2. Manually alter the captured design to take advantage of the new language's semantic constructs. This should be done by a programmer familiar with the new language and the intended design paradigm (e.g., OOA, SSA, etc.).
3. Forward engineer the design to executable source code for the new language using the repository's source code generator

On the other hand, line-for-line translators do enjoy some support. The advantages claimed include:

There is no loss of understanding by the current system's maintenance staff: Extensive loss of comprehension by your system's maintenance staff (as described in the previous section on restructuring pitfalls) does not occur with line-for-line translators. The control flow and data flow of the resulting system should match that of the original system.

An aid to understanding the new language: Since the target language is usually new to the organization, most of the current programming staff will be unfamiliar with it. Line-for-line translators give the programmers a point of reference for understanding the new language by comparing the old language modules with the corresponding, translated modules in the new language.

Consolidating multiple systems to a standard language: Some DoD systems are functionally identical but were written in different languages or dialects. Line-for-line source code translators can operate on each individual language to create a single language which can be compiled by a single compiler. This consolidation could then be followed by other reengineering strategies such as restructuring, design capture, etc.

So depending on your circumstances, line-for-line source code translators may fit your short-term reengineering needs. But if you have the resources, the better approach is the three step process outlined previously—reverse engineer to a repository, manually redesign, and generate executable source code from the new design.

5.9 REENGINEERING PROBLEMS WITHIN THE DoD

There are several problems encountered by software reengineering projects that appear quite often within the Department of Defense. Although not actually unique to the DoD, these problems seem to be aggravated by conditions within the DoD.

The first of these is the penchant for changing military management of software programs. Every few years, the officer in charge of one or more software programs is transferred to another location or project. This creates a discontinuity of key personnel in the midst of a long term reengineering project. Similarly, a high ranking officer or manager may finally be convinced that the commitment of resources (money and personnel) is worth the goals of reengineering when that person is replaced. Thus, the justification process must begin all over again. If you find your organization is prone to replace software project managers periodically, we suggest planning a reengineering project that can be accomplished with the existing personnel (perhaps using the incremental reengineering strategy). Carefully document the reengineering process used and the return on investment (indicating a need for defining and collecting metrics!) to justify the next reengineering project to the new manager.

The next problem prevalent within DoD software organizations is the physical separation between development and maintenance organizations. This tends to encourage what is called the "over the wall" syndrome. That is, the development organization throws a new software application "over the wall" to the maintenance organization with little regard for the maintainability of that application. To complicate this scenario further, the application's users are often physically separated from either development or maintenance organizations. If the maintenance organization reverse engineers their legacy software into a design repository but still receive new applications in the same old format from the development organization, design the return on investment is greatly decreased. The development organization is not using the capabilities of the repository for reuse of software components, tools integration (testing, metrics, graphical display of design, etc.), and other repository features. And we doubt most maintenance organizations will want to continually reengineer all new software applications to fit their new maintenance environment.

Some form of business process reengineering (BPR) needs to occur within the DoD to facilitate communications between maintenance and development organizations. Software applications should be written with future maintainability built in. But for this to occur, the two organizations need to better understand each others needs and requirements. Thus, the practice of physically separating development and maintenance organizations must end.

The last problem that must be overcome within the DoD environment is that of budget control. Typically, the software users control the maintenance budget for their applications. Since software reengineering is primarily a benefit to the maintenance organization, a case must be built to fund a reengineering project using the users' money but without giving the users additional functionality. This can be difficult. The only reason a user organization would fund a reengineering project, without additional functionality, is when the software application has deteriorated to such an extent that the application is unreliable and/or user-requested modifications take a very long time to implement and, when implemented, cause more problems than they solve. One solution to this is the incremental approach to reengineering. Incremental reengineering allows the introduction of new functionality once the software has been validated following reengineering.

5.10 REENGINEERING EMBEDDED OR MCCR SOFTWARE

There are some problems when reengineering embedded or MCCR (Mission Critical Computer Resources) software. This is software that is usually hard coded onto a customized platform. Embedded software can be found in missile guidance systems, weapon systems, radar systems, avionics, etc. Response time is of paramount importance. But so is software dependability. Many of these systems are aging to the point where they need to be replaced or at least upgraded. Most of these systems use a software language that is unique to a given DoD service. Within the USAF, that language is JOVIAL. As might be expected from the preceding discussion, reengineering such systems can present some serious problems. Since the language used does not have a large user base, COTS reengineering tools are not plentiful. Generally, since the language does not have a universal standard definition set, each compiler is different. Thus, the few reengineering tools available must be customized for each compiler.

The target language and execution platform must allow for strict response time constraints. Response time constraints mean a dependence on the hardware platform's properties. Thus, you find a lot of assembler language used within embedded systems. Assembler is a common language but, like JOVIAL, the instruction set varies widely between platforms. So reengineering tools have to be customized.

Often, embedded systems execute on multiple hardware platforms. So even if customized reengineering tools exist for one version of the system, other versions exist that probably include language features different from each other.

There is a significant increase in risk when reengineering embedded systems. The failure of such systems may cause loss of life or severely impact logistical processes. Organizations that reengineer embedded software require a longer schedule due to quality control checks, detailed validation (including redesign of testing platforms), simulations, redesign of the hardware platform, sensor interface considerations, etc. This creates a longer transition period during which two parallel systems must be supported—the legacy system now being used and the newly reengineered system. Methods to implement modifications during the reengineering project become more critical.

5.11 EXECUTION TIME VS. MAINTAINABILITY

There may occur a trade-off between the speed of execution and maintainability. Here again, metrics are important when assessing the response time of a given software application. Whether true or not, users may complain about slow response time following the reengineering of their favorite application. Only metrics before and after the reengineering project will be able to decide whether slower response time is due to the reengineering or whether slower response has occurred at all.

Changes in logic structure, language, or platform can all impact the execution time of the reengineered software. This is a critical issue for embedded applications (weapon systems, navigational systems, radar, etc.). So be careful when reengineering applications where execution speed is of primary importance. MIS application users will only tolerate so much response time degradation for their online, real time applications. Easier maintenance does not necessarily mean slower response time. But if the reengineered software does run slower, then you must decide whether significantly easier maintenance is worth the slower response time.

5.12 SUMMARY

The purpose of this section was not to dissuade you from reengineering. The purpose was to strip away any false hopes that reengineering will quickly and painlessly solve all your organization's maintenance problems. We don't want reengineering to follow the difficult path taken by AI and CASE. Our goal, as change agents, is to manage expectations. Go into reengineering with your eyes wide open.

This Page Intentionally Left Blank

6

REENGINEERING PROJECTS AND SERVICES AT THE STSC

6.1 INTRODUCTION

There are several major software reengineering projects in which the STSC is participating. These include:

SRAH: The Joint Logistics Commander's (JLC) Software Reengineering Assessment Handbook (SRAH) helps to inventory legacy software, select appropriate reengineering strategies, derive costs to reengineer, and aid management in their reengineering decisions.

Reengineering Projects Survey: A major survey of software reengineering projects.

IEEE's Reverse Engineering Newsletter: This quarterly newsletter is published by IEEE's Technical Council on Software Engineering. STSC solicits articles and acts as editor.

STSC Reengineering Technology Report: This bi-annual report represents a synthesis of all the information gleaned from conferences, journals, vendors, and contacts from within the domain of software reengineering

STC's Reengineering Track: Each April the STSC hosts the Software Technology Conference (STC) in Salt Lake City, Utah. The largest annual DoD software conference, STC includes a track on software reengineering. STC '95 also included two tracks on business process reengineering (BPR).

Reengineering and BPR Tools Data Base: This collection of over 400 COTS and GOTS software reengineering tools is supplemented by an additional collection of over 400 software analysis tools. Recently, we've also been collecting information on BPR tools. These tools are categorized (ref. Section 2) and freely available to anybody within the USA.

9-Step Reengineering Preparation Workshop: The STSC has been applying the 9-step preparation process (ref. Section 4) and the SRAH to aid USAF organizations in planning software reengineering projects. This week long workshop helps focus a software organization on what is required for a software reengineering project to succeed.

Introductory Reengineering Tutorial: This half day tutorial introduces the audience to the basics of software engineering including the STSC 9-Step Reengineering Preparation Process, taxonomy, pitfalls, and future directions.

JOVIAL Reengineering Tool Set (JRETS): The development of a JOVIAL Reverse Engineering Tool Set (JRETS) was undertaken by the STSC due to a lack of COTS reengineering tools for the JOVIAL language. Developed under USAF funding, JRETS is now available to anybody requesting it (within the USA).

COBOL Reengineering (COBRE): Based on a comprehensive survey of COBOL usage within the Air Force, the COBRE report includes a list and evaluation of the top COBOL reengineering tools.

6.2 SRAH

STSC's efforts in reengineering strategy selection and cost analysis began with our matrix model appearing in the *1992 Reengineering Tools Report* and later in the March 1992 Crosstalk article "Time to Reengineer?". Thanks to the efforts of the JLC workshop on reengineering (otherwise known

as Santa Barbara-1), and input from numerous reviewers, the model has undergone significant changes and has evolved into *JLC-HDBK-SRAH* or just SRAH (Software Reengineering Assessment Handbook) [SRAH 95].

Throughout 1993, a small sub-group of Santa Barbara-1's panel three continued to meet and refine what was then called the REH (Reengineering Economics Handbook). Aided by two contractors hired by the USAF Cost Agency to coordinate the modifications, a draft version was completed in March of 1994. This draft handbook was called the Software Reengineering Assessment Handbook (JLC-HDBK-SRAH) draft version 1.0. Reviewers suggested modifications for most of the rest of 1994. Based on these suggestions, a final version (version 2.0), was finished by early 1995.

The SRAH consists of three sections: 1) a reengineering strategy selection process, 2) a cost model, and 3) a management decision process.

For the strategy selection process, the matrix model (as proposed by the 1992 *Reengineering Tools Report*) was discarded in favor of a more precise decision model wherein each reengineering strategy has an associated series of multiple-choice questions. Each question has three possible answers. If the average of the summation of all answered questions surpasses a certain threshold, then that reengineering strategy is recommended.

Volume 2 of SRAH contains the reengineering cost models as adapted by their respective agencies. These currently include:

- SEER-SEM
- PRICE-S
- SLIM
- SOFTCOST-OO
- CHECKPOINT
- COCOMO/REVIC²

Once the reengineering strategies have been identified (for the chosen legacy software) and a rough cost estimate established for each, then the findings are presented to management for prioritization and decision.

Thus, each of the three major sections of SRAH is intended for a different audience. The strategy selection section is intended for use by the maintenance support staff. The cost model is to be used by the organization's cost projection personnel. Management uses the output of the previous two sections (strategy selection and cost) to make a decision as to which reengineering strategies, and their associated legacy software system, should be implemented first.

6.3 REENGINEERING PROJECTS SURVEY

To justify the methodology used by the SRAH, an effort was begun at the STSC to collect reengineering project data. Two surveys were developed. The "Pre-Reengineering Survey" is intended for those organizations planning or pursuing a software reengineering project. The "Post-Reengineering Survey" is intended for those organizations that have already completed a software reengineering project. Each survey is accompanied by a set of instructions that explain each question asked. The data from both surveys will be used to validate, and suggest modifications to, the SRAH. Most of the resulting information will be available to the public.

There is a strong need within the DoD for case studies of software reengineering projects. DoD software managers need justification for reengineering projects. Case studies not only demonstrate the advantages of reengineering but also help to identify potential pitfalls—no need to repeat their mistakes.

²Not included in SRAH version 2.0.

The survey data is online at the STSC using a Paradox data base on the DEC local area network. There is some discussion of making the information available on the STSC bulletin board system (BBS). If this occurs, DoD-sensitive information will be deleted along with the names and personal information of those participants that do not wish this information made public.

6.4 IEEE's REVERSE ENGINEERING NEWSLETTER

In spring of 1994, the STSC was asked to take over production of IEEE's *Reverse Engineering Newsletter*, a quarterly publication published by IEEE's Technical Council on Software Engineering (TCSE). The STSC solicits articles and formats the resulting newsletter for publication by IEEE.

To submit an article, send a copy to Michael R. Olsem, editor, at OO-ALC/TISEC, 7278 Fourth Street, Hill AFB, Utah, 84056-5205 or email a copy to olsemm@software.hill.af.mil. To join the distribution list, send an email message to tcse@computer.org with your name, address, and email address.

6.5 THE STSC REENGINEERING TECHNOLOGY REPORT

Volume 1 contains a review of the reengineering domain while Volume 2 contains complete lists of software reengineering tools from the STSC reengineering tools data base. This report is updated every other year so the next report will be published in 1997.

6.6 STC's REENGINEERING TRACK

For the past four years, a track devoted to software reengineering has been included in DoD's Software Technology Conferences (STC). Speakers are chosen who focus on lessons learned from actual software reengineering projects within the DoD. Held every April in Salt Lake City, STC is the largest annual DoD software conference. Over 3,000 attendees and vendors are expected for STC '96.

6.7 REENGINEERING AND BPR TOOLS DATA BASE

The STSC is constantly updating a large database (over 400) of reengineering tools and their vendors. In addition, we also maintain a data base of over 400 software analysis tools. If you have a reengineering project coming up, give us a call and we can give you a free list of software tools that will fit your reengineering project needs.

In addition, the STSC can request vendors to test their tools against any sample software you provide. If you have a reengineering project and you've narrowed the list of tools that can help to the top three or four, we will contact those vendors and have them provide sample output based on your sample code.

Recently, the STSC has begun to collect information on tools that support Business Process Reengineering (BPR). While not nearly as large as the reengineering and analysis tools data base, this effort is growing rapidly.

6.8 9-STEP REENGINEERING PREPARATION WORKSHOP

Based upon the 9-step reengineering preparation process (ref. Section 4) and the SRAH, the STSC conducts an onsite, week-long workshop on planning a reengineering project. By the end of the week, a detailed assessment of the organization's reengineering readiness and a set of reengineering strategies are derived for the legacy software to be reengineered. The subsequent report can be used as the template for the organization's reengineering goals, strategies, and costs.

6.9 INTRODUCTORY REENGINEERING TUTORIAL

Many organizations and conferences are unfamiliar with software reengineering concepts, goals, and strategies. This half-day STSC tutorial teaches the taxonomy, issues, preparation, pitfalls, and future of software reengineering.

6.10 COBOL REENGINEERING (COBRE)

The COBRE project includes a survey of major USAF COBOL installations. This survey identifies the most common hardware platforms, DBMS's, COBOL dialects, etc. within the USAF. The report includes a list and evaluations of the top commercial-off-the-shelf (COTS) COBOL reengineering tools, and lays out a COBOL reengineering environment we feel is optimal given the current state of technology. The survey and its analysis was completed February 1993.

The proposed COBOL reengineering environment is based largely upon existing COTS software. The environment has two possible goals:

- Restructuring the target COBOL system and eliminating dead code, uninitialized variables, etc.
- Translating to executable Ada source code.

The goal of restructuring and conformance to good COBOL coding standards could be implemented for those systems whose remaining support life is relatively short with a high degree of maintenance (i.e., many changes). If the target COBOL system will be around longer, then reengineering radical changes to its source code (i.e., changes greater than 30% of the legacy code) requires translation to Ada according to current DoD regulations.

6.12 SUMMARY

The STSC and the DoD are convinced that reengineering tools and techniques are key to improving the level of maintenance and quality of DoD software. Many services offered by the STSC are largely free. Whatever your reengineering project may need, the STSC can help.

7.1 INTRODUCTION

Most people in the software industry now recognize we are on the brink of a software crisis. The problems of maintenance outlined in Section 1 threaten to overwhelm most data processing organizations. A major justification for the existence of the STSC is to look for solutions to this crisis. But it is dangerous to assume that any new technology, or any combination of technologies, will solve all the problems of maintenance and software quality. Too often in the past, software organizations have jumped on each new technology or methodology as a panacea for all their ills. This quick-fix attitude occurred with CASE, 4GLs, artificial intelligence, and object-oriented design, just to name a few. By itself, reengineering is not a total solution to the problems of software maintenance, but can be viewed as a bridge to that solution.

The solution to the software maintenance crisis is defining and enforcing strict maintenance processes. Reengineering can port your legacy software into this new maintenance environment. But reengineering requires careful planning and careful matching of legacy software to platforms, tools, process, and business goals.

Software maintenance has been described as attempting to change the tires of a moving vehicle. Organizations with large libraries of existing code must simultaneously maintain and continue to run systems critical to their survival. Even for relatively error-free code, organizations still need to modify existing software to use new hardware and to adapt to a rapidly changing, competitive world. Those software vendors that can provide tools or services to ease the pain of maintenance and deliver high quality software stand to do well in the 1990's.

This section describes some of the "hot" new areas of software reengineering. We feel these future directions hold great promise for software reengineering. Some of these areas have COTS tools already while others do not.

7.2 REPOSITORY TECHNOLOGY

Repositories have a long way to go before meeting all the functionality described in Section 1. Repositories will be central to your organization's survival as they begin to incorporate not only software information but enterprise models and business strategies. Business reengineering will create process models of your organization's goods and services. These enterprise models will need to be supported by the organization's software systems. Both enterprise models and software systems will reside and be maintained in the organization's repository.

7.3 PROCEDURE-ORIENTED TO OBJECT-ORIENTED TRANSLATIONS

Some vendors are exploring ways to convert procedure-oriented (i.e., declarative) design into object-oriented design. The obvious problem with such a strategy is the identification and definition of potential objects within the procedural source code. The process includes (see Figure 7-1):

1. Code-slicing

This process locates within the source code everywhere that a given data item is used. If you look at your legacy code, it probably contains high level data fields that are sub-divided into smaller data fields. These high level data fields make a good place to start when deciding what would make good objects. Slicing tools can then display all the processes that use each of the high level data fields. These processes can become "methods" to the data field. We call the resulting data field and associated processes a "proto-object".

2. Externalize tables

Internal tables need to be externalized and turned into objects.

3. Reconcile logic redundancy

Some code logic may be shared between proto-objects. Slicing will display this. The logic may remain redundant within multiple proto-objects or you may want to decide whether both proto-objects should exist or whether one should be defined beneath the other.

4. Add other object-oriented characteristics

The above 3 steps will help you define proto-objects but there are still some extensive manual steps that must be taken to fully define objects. These include strictly defining the relationships between objects, inheritance, etc.

Boeing is using AI principles to translate procedural design to object-oriented design. Their approach uses a series of libraries (application, domain model, and reuse) and rules to extract reusable components. Data records and procedures are separated and analyzed then fed into a state machine containing objects, slots, instances, models, and transition tables. The outputs include the methods and slots of an object-state model.

"Wrappering" is another approach to transforming procedural code into object-oriented code. This involves wrapping an object around a functionally cohesive procedure. Thus, the procedure becomes a method within the object.

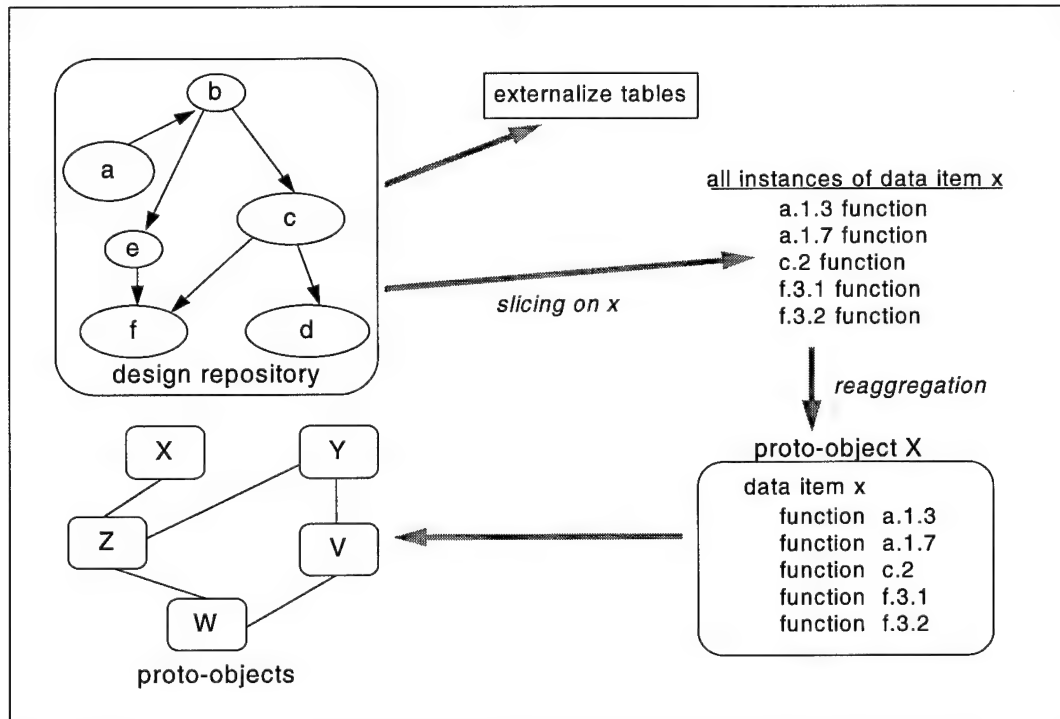


Figure 7-1. Process-oriented to Object-oriented.

At Offutt AFB, COBOL source code was converted into object-oriented Ada code. The following steps are courtesy of Capt. Charles J. Locascio who managed this reengineering project:

1. Reverse engineer to yield data flow diagrams and written information sufficient to explain them. The documentation should include (of course) descriptions of the data sources, sinks, and flows, and of the processes performed upon the data.
2. From these diagrams and documentation, derive a list of functions performed by the system. Draw on the processes identified in your reverse-engineering step for this list of functions. These can be translated into **action-oriented** requirements. Set this preliminary list of requirements aside.
3. At this point, you must change your mindset. **Stop thinking in terms of functional decomposition and switch your perspective to object-orientation.** Perform an analysis of the domain in which the functions are performed. The product of this analysis will be a "concept map"—a conceptual, graphical depiction of the "real-world" entities which make up the domain, and the relationships between them. This analysis should be done without too much regard for which of the entities were represented in the "old" system.
4. Identify the "real-world" entities which are pertinent to your automated system by examining the list of requirements developed in step 2. This step will yield a preliminary list of "objects".
5. Associate each of the requirements in the list developed in step 2 with one of the objects identified in step 4. Remember, an object can be identified as the data which is used to represent an entity plus the operations performed upon that data. Let this brief definition guide you in assigning requirements (which will become operations) to the objects. It is likely you will have requirements which do not seem to fit any of the "real-world" objects.

Consider whether it is appropriate to identify new objects (usually related to interfaces with external systems) which would provide these operations either to the other objects or to the main driver procedure.

6. Analyze your preliminary list of objects and their operations to re-evaluate and refine your list of requirements, and review the requirements carefully with your customers (end-users). Proceed with the other steps of object-oriented analysis as you would in any normal "forward engineering" development. For the most part, you're finished with the products of your reverse engineering, except as references and as aids to maintaining the "old" system while you forward engineer the new one.

7.4 SINGLE-THREAD TO PARALLEL TRANSLATIONS

With connections now using fiber optics, we have reached the theoretical maximum execution speed—the speed of light—on computers using single thread technology. The only way to increase the hardware aspects of CPU performance is by putting circuit elements closer together, finding new materials (such as gallium arsenide circuits), or eliminating any bottlenecks still remaining (e.g., slow gates, multi-step clocks, etc.).

An alternative strategy is to alter the order of software execution by programming for parallel processing. Manufacturers already know how to create computers with millions of chip-size computers - all residing and cooperating within a single processing unit. But our legacy software is almost exclusively single-thread. That is, each process step must wait for all preceding steps to finish before executing. There is a growing need for reengineering tools that can automate the process of identifying and splitting sequentially executing software into components that can run in parallel (i.e., concurrently). Only when such tools are available, will parallel computer manufacturers be able to branch out into the business world the same way they now dominate the scientific world of data processing.

Another aspect to this is reengineering software from mainframe platforms to client-server architecture. Several vendors and the I-CASE project are looking at tools and methods to facilitate this transition. Some of the issues involved include disposition of the data base (centralized versus distributed) and the logical distribution of the executable code modules. For a detailed discussion on using the IEF tool set (Texas Instruments, Inc.) to migrate host applications to client-server, I refer the reader to an article in *CASE Trends* by C.T. Brown entitled "Early Reengineering Tool Sets Focus on Taming Mainframe Beasts" [Brown 93].

7.5 ARTIFICIAL INTELLIGENCE

The technology of Artificial Intelligence (AI) is now being used by many software domains. Subtly, AI is acting as an enabling technology for more and more software tools. The domain of reengineering is no exception. If a central repository is the key to future software development and maintenance AI may become the key to the central repository.

Recognizing that the main purpose of a repository is the storage and retrieval of design specifications, business rules, strategies, and the like, several researchers have suggested such repositories are very close to the functionality of a knowledge base. The repository is a place for the storage of meta-data regarding programs, systems, and data base representations. Maintenance programmers will become knowledge workers as an organization's decision-makers come closer to full control of the products and processes of software evolution [Ulrich 91].

As vendors strive to develop reverse engineering tools that are more powerful, more flexible, and more user-friendly, embedded AI routines are being used as a major competitive advantage. InterCASE from Interport (Fairfax, VA) is an example of an AI-based code reversal tool with its own proprietary, object-oriented repository. Using a knowledge base of rules about programming languages and translation algorithms, InterCASE can generate unique parsers based on a set of grammar rules, semantic actions, and generation routines. By recognizing/storing syntactic and semantic constructs within the source code, they can migrate applications between different hardware/OS platforms [Davis 91].

Researchers at Arthur Andersen's Center for Technology Research are developing a knowledge-based Software Re-engineering Environment (SRE). The design knowledge extracted is stored within a Global Knowledge Base [Davis-3 91].

Computer-Aided Reengineering Software (CARES) depends on knowledge bases for both reverse and forward engineering. Easily customized parsers for different languages can be used interchangeably in a CARES toolset within a CASE environment [Davis-2 91].

One area that has not been addressed by vendors is that of translating rule-based systems into other languages. One of the serious drawbacks to rule-based software is that of execution speed. Due to the nature of the inference engine, and its method of iteratively passing through the entire rule base searching for all possible matching goals, most mid-sized and large rule-based systems are very slow. Special hardware/OS systems are sometimes needed just to execute such AI systems. Alternatively, many expert system developers will model their systems using rules, then manually translate them to a faster language such as C. What is needed are reengineering tools to automate this translation.

7.6 BUSINESS RULES EXTRACTION

Usually, reverse engineering starts from source code and extracts design information. But some tools and techniques are emerging that are beginning to extract business rules from source code and data configurations. It has long been acknowledged that an organization's software systems are valuable, in part, because they represent knowledge (both current and historical) about the organization, its clients, mission, strategy, and implied constraints.

How can a software tool infer an organization's way of conducting business based solely on the existing source code or data record?

If you think about how software is coded, you realize that subtle clues are left behind regarding intent. For example, if a salesman's data record includes room for only three districts, then one might infer that a salesman can only service, at most, three sales districts. When the system was originally developed one can assume the software designer took the time to enquire how many sales districts the company allows per salesman. Even if company policy has changed (or the software developer did not properly execute the user requirements phase), just knowing the software system uses this premise is important for any future maintenance.

Some day, the organization may want to codify the rules by which they conduct business. This may take the form of business reengineering process definition or expert system rules. Perhaps, one day, all software design will be business rule-based to reflect the business environment. Low level

constructs could be combined into high level rules. Reverse engineering tools that help to automate this will aid the organization's management in better understanding the explicit and implicit rules under which the organization's key software systems were written.

Software reengineering should aid the process of business reengineering by extracting components of the legacy software to correspond to business areas. These components could then be re-aggregated to represent a new system that is oriented to support the business processes as defined by business reengineering.

7.7 SUMMARY

Software organizations are clearly challenged by several major trends:

- Change is the only constant for technology, business needs, and requirements
- Elimination of duplication/redundancy of systems
- Conform to standards
- Business reengineering
- Quality assurance
- Declining budgets
- Declining trained personnel requires increasing efficiency
- Open systems
- Concurrent processing
- Reuse
- Ada (for the DoD) and object-oriented design
- CASE
- Software process improvement (SEI's CMM) and information engineering
- The need for software integration of commercial-off-the-shelf (COTS), government-off-the-shelf (GOTS), and in-house development

While reengineering is not the solution to the above, it is a major enabling technology. A maintenance organization's short-term goal is to clear the growing backlog of maintenance demands. The long-term goal is to support change at the requirements level. As organizations begin to think in global terms, their primary challenge will be to use this technology under diverse social and political cultures. The key to success in the 1990's is to position your organization for maximum flexibility to allow for rapid incorporation of new tools, strategies, and methodologies [Ulrich 91]. The basis for this flexibility will be the central repository. And the bridge from existing software systems to repository-based software maintenance is reengineering.

This Page Intentionally Left Blank

APPENDIX A:

REENGINEERING PRODUCTS LIST

A.1 REENGINEERING PRODUCTS LIST BY TYPE

BUSINESS PROCESS REENGINEERING

ANDRULIS/BPre+	MAXIM
ANSWER:Architect	Object Management Workbench (OMW)
Applications Manager (AM)	ParaSET
Arena	PenAnalysis
ART*Enterprise	PenAnalysis Modeling Tools
ASURE	PRIDE Information Factory (PRIDE)
BPSimulator	ProSim
BPwin	PROSLCSE
CLEAR Process	ProVision Workbench
Client/Server Step By Step	Ptech
DBStar	ServiceModel
DECmodel for Windows	SIMPROCESS
Design/IDEF	SmartER
Designer/2000	StP/Booch Method (StP/Booch)
Developer/2000	System Architect (SA)
Developer/2000 for Windows	Transtar Repository
Flowmark	WizdomWorks!
Key for Workgroup	WorkFlow Analyzer (WFA)
KEY:Insight	

DATA REENGINEERING

\$NAME	EXTRACT
ADW/Load (ADW Load)	Foundation Translator
ARRAE	IE: Advantage
ART*Enterprise	Information Systems Engineering Env. (ISEE)
Assembly Design Extractor (ADE)	Integrity Programming Environment
BACHMAN Database Design	InterCycle
BACHMAN/ANALYST	Interleaf 6
BACHMAN/Analyst Capture	M.A.T.I.S.S.E.
BACHMAN/CONVERTER	MacDesigner
BACHMAN/DBA	Mesa/Postscript Support for CASE (Mesa/PS)
CaseConnection/ADW	Mesa/Teamwork Model Bridge (Mesa/TMB)
CLEAR Plus	Open TRANSL8
COBOL Analyst	OpenODB
CQS-Data Engineer (CQS)	Panorama C++
Current Systems Analysis (CSA)	PM/SS
Current Systems Modification (CSM)	Reverse Engineering Tool
Data Dictionary Tree	SQLASSIST
DATAKOM to SQL Conversion Aid	SuperNOVA
DATATEC	System Architect (SA)
DB Data Analyzer	Teamwork/IM
DB Designer	Transition Engineering Facility (TEF)
dbLOADER	TRANSLATION / MIGRATION SERVICE
DBStar	TranZform
DCD III	UniKix
DDS-Link	URMA
Deft CASE System	VAX SCAN (SCAN)
Design Review-Scope	VIA/ALLIANCE
Designer	VIA/Renaissance
Designer/2000	VISION:NorthStar
EasyCASE	Vitro Automated Structured Testing Tool (VASTT)
Environment for Code Re-Engineering (ENCORE)	X-ANALYSIS
ExpressC/Ada	XperCASE

FORWARD REENGINEERING

Activation Framework	Micro Focus Cobol/2 Workbench
Ada Development Environment	MicroSTEP
Ada Software Development Toolset	Migration Workbench
Ada/ADADL Requirements Interface System (ARIS) Model	
AdaFlow	NETRON/CAP
Adagen	Object-Oriented Structured Design/C++
Ada_Z	ObjectMaker
ANSWER:Architect	OMTool (OMT)
Applications Manager (AM)	ONTOS VIA/OIS
ARIS	Open Interface
Autocode	OpenODB
BattlePlan	Panorama C++
BPwin	Panorama C++/OO-Browser
C Development Environments	ParaSET
C++ Development Environment	PARITY
CA-MetaCOBOL+	PC DICTIONARY
CA-TELON	PowerDesigner
CA-TELON PWS	PowerTools
CASE Designer	Predict
case/ap	PRIDE Information Factory (PRIDE)
COBOL Program Generator (CPG)	PRODOC re/NuSys Workbench
Code Generator	Rational Rose (Rose)
Composer	Recycle-SF/Repair
Cradle SEE	ReEngineer
Current Systems Analysis (CSA)	REENgineering Environment & Workbench (REENEW)
Current Systems Modification (CSM)	Reverse Engineering Tool
DAISys	SmartChart
Data Dictionary Tree	SMARTsystem
DATATEC	SNiFF+
DBStar	Software Reengineering Environment (SRE)
DDS-Link	Software Refinery
Design Recovery Series	STARK
EasyCASE	superCASE (SCI)
EiffelCase	SUPRe/DAISys
Ensemble	Teamwork/Ada Source Builder
Ensemble Viewer	Teamwork/C Source Builder
EPOS	Teamwork/OOD
ERwin	Technical Systems Engineering Environment (TSEE)
Excelerator for Design Recovery	TeleUSE
FORCE	The Migrator Work Bench (Migrator)
FORTRANgen	Translator
Foundation Vista10	Tree4(C, Fortran, Pascal)
FreeFlow	UNISSET
Huron	ViewCenter
IE: Advantage	VIEWS-SF
Information Engineering Workbench (IEW)	Vitro Automated Structured Testing Tool (VASTT)
Information Systems Engineering Env. (ISEE)	Workflow Analyzer (WFA)
IntegrAda	X-ANALYSIS
LANSA	XAda
LBMS SYSTEMS ENGINEER	Xinotech Program Composer
LOOK & FEEL	XperCASE
METAgen	

Appendix A: Reengineering Products List

REDOCUMENTER

ABSTRACT/PROBE+	Maintenance Workbench
Ada Design and Documentation Language (ADADL)	Military Standard Document Conversion (MSDC)
Ada Development Environment	Navigator
Ada Software Development Toolset	OBJECT IE
Ada-ASSURED	Object-Oriented Structured Design/C++
AdaFormat	Panorama C++
ADAPRETY	Panorama C++/OO-Analyzer
AdaReformat	Panorama C++/OO-Browser
Automated Documentation System (ADS)	Panorama C++/OO-Diagrammer
Blue Four	PC DICTIONARY
C Design and Documentation Language (CDADL)	plusFORT
C Development Environments	PM/SS
C-DOC	PRIDE Information Factory (PRIDE)
C-LIST	PRODOC re/NuSys Workbench
C-Vision for C	Q/Artisan
CA-MetaCOBOL+	Q/ARTISAN PC
COBOL Analyst	QualityFirst C
COBOL spII	RE-DOC
COBOL Structuring Aid (CSA)	RE-SPEC
CONFIGURE	ReEngineer
DATATEC-DS	REengineering APplications (REAP)
DCD III	REENGINEERING Environment & Workbench (REENEW)
DEC FUSE	Refine/Ada
DOCBUILDER	Refine/C
DocEXPRESS	Refine/Cobol
DOCGEN/C	Refine/Fortran
Document Generator (DocGen)	SCAN/COBOL
Documentation-Aid, MVS (Doc-Aid)	SMARTsystem
Documentor	Software Document Automation (SoDA)
DOSSIER BROWSE	Software Refinery
DOSSIER PROVE	Source/RE
Eagle	STATEMATE
EPOS	Structure & Logic Analysis Module
EPOS 2000 (EPOS 90)	SUPERSTRUCTURE / RETOOL
ESW Documentation	Teamwork/DocGen
Excelerator for Design Recovery	Teamwork/IM
F-SCAN	TeleUSE
FORTRANgen	Translator
FOR_STRUCT	Tree4(C, Fortran, Pascal)
FOR_STUDY	VAX SCAN (SCAN)
Foundation Vista10	VIA/ALLIANCE
HP Ada/300 Development System	VIA/SmartDoc
Information Systems Engineering Env. (ISEE)	Visible Analyst Workbench (VAW)
IntegrAda	VISION:Legacy
MacDesigner	Xinotech Program Composer

RESTRUCTURER

AdaSplit	PRODOC re/NuSys Workbench
ADW/Recoder	Q/Artisan
BACHMAN Database Design	Q/ARTISAN PC
Blue Four	QA C
C-DOC	QA C++
C-LIST	Reformat
COBOL Structuring Aid (CSA)	Requirements Driven Development (RDD)
COBOL Structuring Facility (COBOL/SF)	Retrofit
COBOL/METRICS	RevAda/StP
CodeBreaker	Reverse Engineering Tool
CQS-Data Engineer (CQS)	SILVERRUN-ERX
DEC FUSE	SmartStar
Environment for Code Re-Engineering (ENCORE)	Software Reengineering Environment (SRE)
Fortran Development Tools	Software Refinery
FOR_STRUCT	SPAG
Logiscope	The Data Design Facility (DDF)
Maintainer's Assistant	TRANSLATION / MIGRATION SERVICE
Mozart	VAXset/Program Design Facility (VAXset/PDF)
Object Oriented Tool (OOT)	Visible Analyst Workbench (VAW)
Panorama C++/OO-Browser	VISION:Legacy
plusFORT	Vitro Automated Structured Testing Tool (VASTT)
PM/SS	X-ANALYSIS

Appendix A: Reengineering Products List

RETARGETING

ANSWER:Zim	Open BASIC
Assembly Design Extractor (ADE)	Open TRANSL8
BACHMAN/Analyst Link	Q/Artisan
Computer Tester Analyzer Controller (C-TAC)	Q/ARTISAN PC
CONVERSION ENGINE	Refine/Ada
Current Systems Analysis (CSA)	Refine/C
Current Systems Modification (CSM)	Software One Exchange
DBStar	Software Reengineering Environment (SRE)
DCL8	SYNTran
Excell 930	TRANSLATION / MIGRATION SERVICE
EXTRACT	TranZform
Foundation SQL	UniKix
Foundation Translator	VADScross
Integrated Design Automation System (IDAS)	VADSmp
InterCASE KnowledgeWare Gateway (IKG)	VADSelf
JOVIAL Reverse Engineering Toolset (JRETS)	VADSWorks
Mesa/CASE Tool Interface (Mesa/CTI)	VAX SCAN (SCAN)
Open ACCLIM8	

REVERSE ENGINEERING

Ada Design & Documentation Language (ADADL)	CodeCheck
Ada Development Environment	Computer Tester Analyzer Controller (C-TAC)
Adagen	CONVERSION ENGINE
AdaMagic	Cradle SEE
AdaQuest	DataModeler
ADW/Load (ADW Load)	DBStar
ADW/Pinpoint	DCD III
Aide-De-Camp (ADC)	DEC FUSE
allCLEAR	DecisionVision 1 (DV1)
Application Browser	Design Recovery Series
ARC SADCA	Design/IDEF
Architecture Design & Assessment System (Adas)	DOSSIER PROVE
AS/SET	EAGLESCAN
ASA 20/20	EasyCASE
AsmFlow Professional	EasyCODE
Assembly Design Extractor (ADE)	EiffelCase
AutoAnalyzer	Ensemble
AutoDiagrammer	Ensemble Viewer
AutoFlow	EPOS
AutoStructureChart	ER-Designer
BACHMAN/ANALYST	ERwin
BACHMAN/Analyst Capture	Excellerator for Design Recovery
Battlemap Analysis Tool (BAT)	EXDIFF
BattlePlan	Existing Systems Workbench (ESW)
BPwin	Expert Debugging Software Assistant (EDSA)
C Design and Documentation Language (CDADL)	ExpressC/Ada
C Development Environments	FORCE
C++ Development Environment	FORTTRAN Reverse Eng & Document System (FREDoc)
C-CALL	FORTTRAN-lint
C-LIST	FORTTRANgen
C-SCAPE	FORWARN
CASE Designer	Foundation SQL
Chen Workbench	Foundation Translator
CLEAR Plus	Foundation Vista10
CMS-2 Design Analyzer (DESAN)	FreeFlow
CMS-2 Reverse Engineering Technology	GILES
COBOL Analyst	GPSA
COBOL Magic	GrafBrowse
COBOL-lint	Hindsight
CodeBreaker	Huron

REVERSE ENGINEERING (CONTINUED)

Information Engineering Workbench (IEW)	Recycle-SF/Recover
Information Systems Engineering Env. (ISEE)	Recycle-SF/Repair
InterCASE KnowledgeWare Gateway (IKG)	Reeng. Doc./Migrator - Source to CASE (RDMSCC)
InterCycle	ReEngineer
J73 Automated Verification System (J73AVS)	REEngineering Environment & Workbench (REENEW)
JCL/Convert	Refine/Ada
JOVIAL Analysis and Conversion Kit (JACK)	Refine/C
JOVIAL Reverse Engineering Technology (JRET)	Refine/Cobol
JOVIAL Reverse Engineering Toolset (JRETS)	Refine/Fortran
KEY:Insight	Requirements & Traceability Management (RTM)
LabVIEW	Requirements Driven Development (RDD)
LANSA	RevAda/StP
Logiscope	Reveng
Logiscope Graphical Editor	Revengg
LOV/OMT	Reverse Engineering Tool
LYDDIA	ROCHADE
Maintainer's Assistant	ROS/ADA
METAgen	SCAN/COBOL
Model	SchemaGen
Navigator	See-Ada
OBJECT IE	SEER
Object Management	SILVERRUN-ERX
Object Management Workbench (OMW)	SIMPROCESS
Object-Oriented Structured Design/C++	SmartChart
Objectivity/DB	Smarter
ObjectMaker	SMARTgraph
OLGA	SMARTsystem
ONTOS VIA/OIS	SNiFF+
Open Interface	Software Reengineering Environment (SRE)
Pacbase with Pacreverse	Software Refinery
PACREVERSE	Software through Pictures (StP)
Panorama C++	Source Code Analyzer (SCA)
Panorama C++/OO-Analyzer	Source/RE
Panorama C++/OO-Browser	SPAG
Panorama C++/OO-Diagrammer	StP/Booch Method (StP/Booch)
Panorama C++/OO-SQA	superCASE (SCI)
Paradigm Plus	System Engineer
ParaSET	Teamwork/Ada
PATHVU	Teamwork/ASG Builder
PC DICTIONARY	Teamwork/C Rev
Persistence	Teamwork/FORTRAN Rev
PLASMA	The Data Design Facility (DDF)
PM/SS	The Migrator Work Bench (Migrator)
POET	Toolbus
PowerDesigner	Transition Engineering Facility (TEF)
PowerPDL	Translator
Predict	Tree4(C, Fortran, Pascal)
PRODOC re/NuSys Workbench	Universal Network Architecture Services (UNAS)
Ptech	URMA
QA C	VAXset/Program Design Facility (VAXset/PDF)
QA C++	VIA/Insight
QA FORTRAN	VIA/Renaissance
QualityFirst C	VIEWS-SF
Rational Rose (Rose)	VISION: NorthStar
RE-DOC	Visual Interactive FORtran (VIFOR)
Re-engineering MENTOR	Vitro Automated Structured Testing Tool (VASTT)
Re-Engineering Product Set	WinScope
RE-SPEC	Workflow Analyzer (WFA)
RE/Cycle	X-ANALYSIS
RE/CYCLE	XperCASE
Re/Toolkit	XRAY Source Explorer
Recycle-SF	

Appendix A: Reengineering Products List

SOURCE CODE TRANSLATOR

ACE*Tester	Metamorphosis
Ada9X Transition Aid	Model
AdaMagic	MPS/400
ARC SADCA	ObjectCenter
Auto-G Case Toolset (Auto-G)	OmniMark
Automated Source Code Conv. Sys. for Ada (ASCCSA)	Open BASIC
AXI	Pastran
CA-MIGRATE/COBOL	PR:QA C (PR:QA C)
CMS-2 Translate to Ada (TRADA)	PR:QA C++
Codewright	PRODOC re/NuSys Workbench
Computer-Aided Software Translator (CAST*)	PROMULA.FORTRAN
CONVERSION ENGINE	Re-Engine Documentor/Migrator
DACS Ada 95	RE-SPEC
DACS DARTS (DARTS)	REENgineering Environment & Workbench (REENEW)
DECset	Software Refinery
Environment for Code Re-Engineering (ENCORE)	STARK
Evaluator	SuperNOVA
FORCE	T-Windows
FORGE Explorer/Browser	The Migrator Work Bench (Migrator)
FORTRAN 77 Programmer's Assistant III	Translate-CGM
Fortran Development Tools	Translation Services
FOR_C	TRANSSLATION/MIGRATION SERVICE
FOR_C++	Translator
Fx Debugger	Turbo-to-C Tools
Hypersoft Application Browser (HAB)	UIL/Trans
JOVIAL Analysis and Conversion Kit (JACK)	Xinotech Language Translator (XLT)
JOVIAL/ADA TRANSLATOR	Xinotech Program Composer
MACYACC	

A.2 REENGINEERING PRODUCTS LIST BY NAME

See Volume 2.

A.3 REENGINEERING PRODUCTS LIST BY VENDOR

See Volume 2.

APPENDIX B:

REENGINEERING PRODUCTS SHEETS

See Volume 2.

This Page Intentionally Left Blank

APPENDIX C:

REENGINEERING PRODUCTS CRITIQUES

See Volume 2.

This Page Intentionally Left Blank

APPENDIX D:

AVAILABLE TRAINING AND CONFERENCES

Title: "Software Technology Conference
Sponsored By: HQ USAF and The STSC
Phone: (801) 777-7411

Title: "Annual Oregon Workshop on Software Metrics"
Sponsored By: Oregon Center for Advanced Technology Education, Portland State University
Phone: (503) 725-3108

Title: "Carma McClure: Reuse Engineering"
Sponsored By: Extended Intelligence, Inc.
Phone: (312) 346-7090

Title: "Reengineering Cobol Applications"
Sponsored By: SEEC, Inc.
Phone: (412) 682-4991

Title: "Reverse Engineering Forum"
Sponsored By: Progress Software Educational Sources
Phone: (617) 280-4560

Title: "Software Re-engineering: Methods, Technologies and Tools"
Sponsored By: Digital Consulting Inc.
Phone: (508) 470-3880

Title: "Reverse Engineering" and "Structuring Maintenance"
Sponsored By: McCabe & Associates
Phone: (301) 596-3080 or (800) 638-6316

Title: "Systems Re-engineering Workshop"
Sponsored By: Naval Surface Warfare Center
Phone: (301) 394-5099

Title: "The National Software Re-engineering & Maintenance Conference"
Sponsored By: DCI, CASE Trends, Keane, Texas Instruments, Oracle, Ernst & Young
Phone: (508) 470-3880

Appendix D: Available Training and Conferences

Title: "SEI Conference on Software Engineering Education"

Sponsored By: SEI, ACM, IEEE

Phone: (412) 268-3007 or 5800

Title: "Conference on Software Maintenance"

Sponsored By: IEEE and IEEE Computer Society

Phone: (202) 371-1013

Title: "Software Reengineering" and "Software Repository and Bridge Technology" and "Impact Analysis"

Sponsored By: Software Evolution Technology (SEVTEC)

Phone: (703) 450-6791

Title: "Adventures in Reengineering" and "Reengineering in the Real World"

Sponsored By: Technology Innovations

Phone: (510) 820-9448

Title: "TSRM: The Systems Redevelopment Methodology"

Sponsored By: James Martin Company

Phone: (703) 620-9504

Title: "Business Process Reengineering"

Sponsored By: Learning Tree International

Phone: (800) 843-8733

APPENDIX E:

REFERENCES AND RECOMMENDED READINGS

- [Army 91] Army, Department of the, U.S. Army Information Systems Engineering Command, Corporate Integration Division, Systems Integration Directorate, *Software Redesign Decision Making*, Contract DAEA18-89-D-0015, Delivery Order 0098, HQ USAISC, Fort Huachuca, AZ, December 9, 1991.
- [Arnold 93] Arnold, Robert S., *Software Reengineering*, IEEE Computer Society Press, Los Alamitos, CA, 1993.
- [Basili 91] Basili, Victor R., and John D. Musa, "The Future Engineering of Software: A Management Perspective," *IEEE Computer*, September, 1991, p. 95.
- [Basili 89] Basili, Victor R., "Software Development: A Paradigm for the Future," *IEEE Computer*, 1989.
- [Brewin 91] Brewin, Bob, quoting Paul Strassmann, "Corporate Information Management White Paper," *Federal Computer Week*, September 1991, p. 10.
- [Brown 93] Brown, C.T., "Early Reengineering Tool Sets Focus on Taming Mainframe Beasts," *CASE Trends*, November 1993.
- [Bush 88] Bush, Eric, "A CASE for Existing Systems", *Language Technology White Paper*, Salem, MA, 1988.
- [Chikofsky 90] Chikofsky, Elliot J. & James H. Cross II, "Reverse Engineering & Design Recovery: a Taxonomy", *IEEE Software*, pp. 13-17.
- [Chikofsky 92] Chikofsky, Elliot J., "Third Reverse Engineering Forum", Northeastern University, Burlington, MA, September 1992.
- [Connell 92] Connall, Duncan G., "An Automated Methodology for Identifying and Naming Data Objects", Global Software, Inc. September 1992., (Presentation to the Third Reverse Engineering Forum, Northeastern University, Burlington, MA).
- [Conner 82] Conner, Daryl R., and Robert W. Patterson, "Building Commitment to Organizational Change," *Training and Development Journal*, Vol 36, Number 4, April, 1982, pp. 18-30.
- [Davis 91] Davis, Jack M., *A Manager's Guide to Software Maintenance & Re-engineering Tools*, version 1.0, Software Productivity Group, Inc. Shrewsbury, MA, May 1991.
- [Davis-2 91] Davis, Jack M., "Technologies of Re-engineering Tools", *Technical Support*, June 1991, pp. 30-35.
- [Davis-3 91] Davis, Jack M., "Software Re-engineering: Capture Tools", *CASE Trends*, Fall 1991, pp. 30-34.
- [DoD 91] Boehm, Barry, *Draft Department of Defense Software Technology Strategy*, September 1991. Prepared for the Director of Defense Research and Engineering (DDR&E) in Partial Fulfillment of the DDR&E Software Action Plan.
- [Fowler 88] Fowler, Pricilla, and Stan Przybylinski, *Transferring Software Engineering Tool Technology*, IEEE Computer Society Press, Washington D.C., 1988.

Appendix E: References and Recommended Readings

- [Hammer 91] Hammer, Michael, "The Last World," *Computerworld Premier* 100, September, 1991, p. 80.
- [Hammer 93] Hammer, Michael and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*, Harper Business, New York, 1993
- [Hampton 91] Hampton, John, Gregory T. Daich, and Debbie Dyer, *Development of a Prototype JOVIAL Re-engineering Toolset*, SAIC white paper report to the USAF, San Diego, CA, 18 September 1991.
- [Harrison 90] Harrison, Warren, and Curtis Cook, "Insights on Improving the Maintenance Process Through Software Measurement", *Proceedings of the Conference on Software Maintenance 1990*, IEEE Computer Society Press, Los Alamitos, CA, 26-29 November 1990.
- [Hammer 93] Hammer, M. and J. Champy, "Reengineering the Corporation: A Manifesto for Business Revolution", Harper Business, New York, 1993.
- [Jacobsen 91] Jacobsen, Ivar and Fredrik Lindstrom, OOPSLA 1991 Proceedings, pp. 340-350.
- [JLC 92] Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management, First Software Reengineering Workshop, "Back to the Future Through Reengineering", Santa Barbara, CA, September 1992.
- [McCabe & Butler 89] McCabe, Thomas J., and Charles W. Butler, "Design Complexity Measurement and Testing", *Communications of the ACM*, Volume 32, No. 12, December 1989.
- [McClure 90] McClure, Carma L., *The Three Rs of Software Automation: Re-engineering, Repositories, Reusability*, Extended Intelligence, Inc., Chicago, IL, 1990.
- [Meredith 91] Meredith, Denis, "Software Re-engineering Alternatives", *Proceedings of the 8th International Conference and Exposition on Software Maintenance & Re-engineering*, Washington, DC, USPD, Silver Spring, MD, 5-9 August 1991.
- [Moad 90] Moad, Jeff, "Maintaining the Competitive Edge", *Datamation*, 15 February 1990, pp. 61-66.
- [Mosemann 92] Mosemann, Lloyd K., "Ada: Vital to the Industrial Base," Address at the *Ada's Success in MIS: A Formula for Progress Symposium*, George Mason University, Fairfax, Virginia, January, 14, 1992.
- [NSWC 92] Naval Surface Warfare Center, "Systems Reengineering Technology Workshop", Silver Spring, MD, August 1992.
- [Osborne 90] Osborne, Wilma M., and Elliot J. Chikofsky, "Fitting Pieces to the Maintenance Puzzle", *IEEE Software*, January 1990, pp. 11-12.
- [Programming Languages Policy 5.c, 7 Aug 90] Mosemann, Lloyd K., "Air Force Policy on Programming Languages - Action Memorandum." Memorandum for the Vice Chief of Staff; Major Command, Separate Operating Agency, and Direct Reporting Unit Commanders; and Air Force Program Executive Officers.
- [Samuelson 90] Samuelson, Pamela, "Reverse-Engineering Someone Else's Software: Is It Legal?", *IEEE Software*, January 1990, pp. 90-96.
- [Sayani 91] Sayani, Hasan, "Payoff of Reverse Engineering into an Open Repository", September 1992., (Presentation to the Third Reverse Engineering Forum, Northeastern University, Burlington, MA).
- [Seymour 92] Seymour, Patricia L., "Re-engineering in the Real World", Technology Innovations, (Tutorial at the Conference and Exposition on Software Maintenance and Re-engineering, Washington DC, June 1992).

-
- [Sittenauer 92] Sittenauer, Chris and Michael R. Olsem, "Back to the Future Through Reengineering: The Santa Barbara I Reengineering Workshop", *CrossTalk*, November 1992, pp. 14-16.
- [Sittenauer-2 92] Sittenauer, Chris and Michael R. Olsem, "Critiques of Reengineering Handbook Sought", *CrossTalk*, December 1992.
- [Suydam 87] Suydam, William, "CASE Makes Strides Towards Automated Software Development", *Computer Design*, 1 January 1987, pp. 49-70.
- [Ulrich 91] Ulrich, William M., "The Re-development Framework: The Transformation Stage", *CASE Trends*, Summer 1991, pp. 24-30.

STSC Technology Reports:

- [CM 94] Software Technology Support Center, *Configuration Management Report*, 1994.
- [PT 94] Software Technology Support Center, *Process Technology Report*, (Volume I, 1994, Volume II, 1995).
- [PME 95] Software Technology Support Center, *Project Management/Estimation Report*, 1995.
- [RE 95] Software Technology Support Center, *Reengineering Technology Report*, Volumes I and II, 1995.
- [RED 95] Software Technology Support Center, *Requirements Engineering and Design Report*, 1995.
- [SEE 94] Software Technology Support Center, *Software Engineering Environments Report*, 1994.
- [TEST 94] Software Technology Support Center, *Software Test Technologies Report*, 1994.

This Page Intentionally Left Blank

APPENDIX F:

GLOSSARY

Ada	The software language mandated for all new U.S. Department of Defense software projects (ANSI/ISO/IEC-8652:1995).
AI	Artificial Intelligence is an umbrella term that covers the fields of expert systems, neural nets, robotics, computer vision, computer speech recognition, and other technologies.
CARES	Computer-Aided Reengineering Software represents the class of tools that aid and automate the process of software reengineering.
CASE	Computer-Aided (or -Assisted) Software Engineering is a set of tools used to implement the discipline of software engineering throughout the development life cycle.
CDIF	The CASE Data Interchange Format is a common data exchange format between CASE tool sets.
COBOL	A computer language generally used in MIS applications.
COCOMO	The COConstructive COSt MOdel is a widely accepted methodology for measuring the potential cost of a proposed software project.
COTS	Commercial off the shelf.
CMM	Capability Maturity Model developed by the Software Engineering Institute at Carnegie Mellon University, Pittsburgh, PA. This model has 5 tiers that represent where a software organization is assessed regarding their software development/maintenance environment.(Technical Report CMU/SEI-87-TR-23, ESD-TR-87-186).
Cyclomatic Complexity	The maximum number of linearly independent paths through a module of code (see "Design Complexity Measurement and Testing" by McCabe and Butler, <i>Communications of the ACM</i> , December 1989).
data name rationalization	Uniform naming of the same logical data item across all software products.
data reengineering	The examination and alteration of an existing data file or data system to reconstitute it in a new form. The process encompasses a combination of subprocesses (as applied to data) such as reverse engineering, restructuring, redocumentation, translation, forward engineering, and retargeting.
DBMS	Data Base Management System.
DITSO	Defense Information Technology Services Organization.
E-R Diagram	An Entity-Relationship Diagram is a means of graphically displaying the relationships and attributes of data objects for the object-oriented analysis methodology.

Appendix F: Glossary

Essential Complexity	A measurement of the degree to which a module contains unstructured constructs (see "Structured Real-Time Analysis and Design" by McCabe et. al., <i>IEEE COMPSAC-85</i> , October 1985).
forward engineering	The set of engineering activities that consume the products and artifacts derived from legacy software and new requirements to produce a new target system.
GOTS	Government off the shelf. Used to denote available government software tools such as JRETS.
I-CASE	Integrated CASE tool environment. A DoD standard for development, maintenance, and reengineering tools that can pass data between tools.
JCL	Job control language. This is the command language that instructs IBM computers on the control flow between programs within the same software system.
JLC	Joint Logistic Commanders.
JOVIAL	A software language for real time systems developed for the U.S. Air Force (Mil-Std-1589A), now largely superseded by the Ada language standard.
JRETS	JOVIAL ReEngineering ToolSet developed by the STSC.
legacy software	Existing production software.
maintenance	The modification of a software product, after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.
meta-	A higher level of abstraction that generally produces a logical model meant to represent a real-world construct. For example, a program is a meta representation of a real-world process such as accounting. Meta-data of design information is the rules that represent the program design information.
MIS	Management Information Systems. Generally all non-real time systems. These would include applications such as payroll, inventory, personnel, accounting, etc.
OOA/OOD	Object-oriented analysis and object-oriented design. A paradigm for software development based on data constructs with associated processes (i.e. methods).
product sheet	A product sheet was solicited from every vendor whose tool appears on our list. The information from those that responded can be found in Appendix B.
RDB	Relation data base. A table driven DBMS.
redocumentation	The process of analyzing the system to produce support documentation in various forms including users manual and reformatting the systems' source code listings.
reengineering	The examination and alteration of an existing subject system to reconstitute it in a new form. The process encompasses a combination of subprocesses such as reverse engineering, translation, restructuring, redocumentation, forward engineering, and retargeting.
reformatting	Pretty printing to make source code indentation, bolding, capitalization, etc., consistent. Considered a sub-domain of redocumentation.
repository	A storage site for software systems' design information.

restructurer	A software tool that makes source code more understandable by implementing modern programming constructs.
restructuring	The engineering process of transforming the system from one representation form to another at the same relative abstraction level, while preserving the subject system's external functional behavior.
retargeting	The engineering process of transforming and hosting or porting the existing system in a new configuration.
reverse engineering	The engineering process of understanding, analyzing, and abstracting the system in another form or higher level of abstraction.
ROI	Return on investment.
SDSA	Software development support activity.
SEI	Software Engineering Institute located in Pittsburgh, PA on the Carnegie-Mellon University campus.
SLOC	Source lines of code.
spaghetti code	Source code designed and written without benefit of a design methodology such as structured or object-oriented design.
SRAH	Software Reengineering Assessment Handbook. A JLC document that helps select appropriate reengineering strategies, estimate cost, and prioritize legacy software to be reengineered.
SSA/SSD	Structured Systems Analysis/Structured Systems Design.
SSC	USAF Standard System Center.
STC	Software Technology Conference. Held every April in Salt Lake City (Utah), this software conference is sponsored by the Army, USAF, Navy, and Marine Corps.
STSC	The Software Technology Support Center is a branch of the US Air Force located at Hill AFB, Utah. One of its goals is to classify and evaluate software tools and methodologies for the purpose of consultation to all US Air Force software development and support activities.
stubbing out	The process of using dummy call routines, or comments, to replace source code, DBMS calls, etc. that are incompatible with your reengineering tools.
source code translators	Transformation of source code from one language to another or from one version of a language to another version of the same language (e.g. going from COBOL-74 to COBOL-85).
systems engineering	The top level process of engineering a system to meet overall requirements.
TCSE	Technical Council on Software Engineering

This Page Intentionally Left Blank

1.1 THE SOFTWARE TECHNOLOGY SUPPORT CENTER

The mission of the Software Technology Support Center (STSC) is to transition technologies and exchange information to help DoD Software Development and Support Activities (SDSA) continuously improve their software quality and life-cycle productivity.

A planned approach is necessary for successful transition. In general, transitioning effective practices, processes, and technologies consists of a series of activities or events that occur between the time a person encounters a new idea and the daily use of that idea. Conner and Patterson's Adoption Curve [Conner 82], shown in Figure 1-1, illustrates these activities.

After encountering a new process or technology, potential customers of that technology increase their awareness of its usage, maturity, and application. If the process or technology is promising, customers then try to better understand its strengths, weaknesses, costs, and applications. These first activities in the Adoption Curve require a significant amount of time.

Next, the customer evaluates and compares the processes and technologies that show the most promise. To reduce the risk, customers usually try new processes or technologies on a limited scale through beta tests, case studies, or pilot projects. A customer then adopts processes or technologies that prove effective. Finally, refined processes and technologies become essential parts of an organization's daily process (institutionalization).

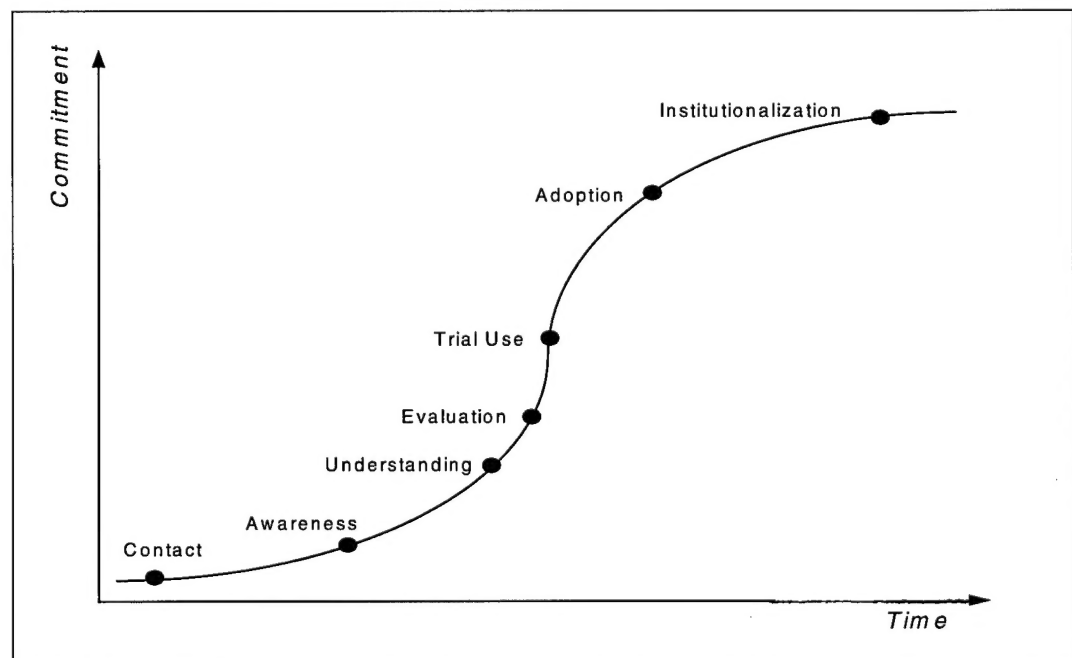


Figure 1-1. Adoption Curve.

Word processors are essential in most organization's daily operations. Yet, 30 years ago they did not exist. The institutionalization of word processors in many organizations followed a series of events similar to those identified in the Adoption Curve.

The STSC is researching and collecting information about technologies that will reduce the time and resources it takes to become aware, understand, evaluate, test, try, and adopt effective practices, processes, and technologies. The STSC has developed the following objectives to accomplish its mission:

- **Technology Evaluation** – Identify, validate, classify, and evaluate effective processes and technologies.
- **Information Exchange** – Facilitate the exchange of better software business practices, processes, and technologies within the DoD.
- **Insertion Projects** – Analyze and improve processes, adopt new methodologies as needed, evaluate and select effective tools, receive appropriate levels of training, and perform pilot projects to try out and confirm the technology insertion efforts.
- **STSC Associates** – Develop STSC associates who can infuse effective process and technology improvements through the use of STSC products, services, and processes.

1.2 STSC TECHNOLOGY TRANSITION APPROACH

This section describes the STSC's approach to the objectives identified in the previous section.

1.2.1 TECHNOLOGY EVALUATION

The first technology transition objective involves identifying, validating, and classifying processes, methods, and technologies that can potentially improve the quality or productivity of software development and maintenance. Many organizations focus on deadlines and customer needs and therefore lack the resources and time to thoroughly investigate options for improvement, leaving them vulnerable to marketing hype. The STSC has developed the infrastructure to provide information on all types of applicable technologies. Product critiques, which are essentially brief evaluations from experienced technology users, are collected. Quantitative evaluations, which are detailed, comparable, and objective, are performed on the most promising tools, methods, or processes.

1.2.2 INFORMATION EXCHANGE

This technology transition objective exposes potential customers to available technologies and, conversely, customer requirements to technology developers. Referring to the adoption curve, this objective focuses on contact, awareness, and understanding. STSC products that accomplish this objective include *CrossTalk*, *The Journal for Defense Software Engineering*, the annual Software Technology Conference, specific technology reports, and electronic customer services.

- **CrossTalk** – Over 16,000 software professionals receive *CrossTalk* monthly. This publication provides a forum to exchange ideas. Articles cover leading edge, state-of-the-art, and state-of-the-practice processes and technologies in software engineering.
- **Software Technology Conference** – The annual Software Technology Conference is held each April in Salt Lake City, Utah. This conference brings together over 2,500 software professionals from government, industry, and academia to share technology solutions and exchange ideas and information.
- **Technology Reports** – STSC technology reports provide detailed information on specific software engineering technologies; this report is an example. The current list of reports includes: *Software Test Technologies*, *Documentation*, *Project Management and Software Cost Estimation*, *Requirements Analysis and Design*, *Reengineering*, *Process Technologies*, *Software Engineering Environments*, and *Software Configuration Management*.

These reports provide awareness and understanding of each topic in preparation for evaluation and selection of corresponding technologies. Over 55,000 of these reports have been distributed. In addition to the technology reports, the following products are available: *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, *Metrics Starter Kit and Guidelines*, and *Cleanroom Pamphlet*.

- **On-Line Services** – The STSC provides electronic access for the software engineering community to information via its On-Line Services. These services include: Bulletin Board System, World Wide Web Home Page, Lynx Browser, Gopher Client/Server, and Anonymous FTP site. If there are questions or comments on these services, or if a company or individual wishes to share information with the software engineering community, please contact the STSC at: OO-ALC/TISEB, attn: George A. Klipper, Information Manager/BBS, 7278 4th Street, Hill AFB, UT 84056, Phone: (801) 777-9712, DSN 777-9712, Fax: (801) 777-8069, Email: klipperg@software.hill.af.mil.
 - **Telnet Connection:** Individuals with Internet capability can connect to the STSC On-Line Services Bulletin Board System (BBS) with the command: `telnet bbs.stsc.hill.af.mil`. When connected, follow the instructions provided to bring up the Main Menu.
 - **Dial-in Connection:** Individuals lacking Internet capability can connect to the STSC On-Line Services Bulletin Board via modem. Dial 801-774-6509 or DSN 775-3602. Set your device to VT emulation. Set your modem to between 2400 and 9600 bits per second, 8-bit word, no parity, and 1 stop bit. When connected, follow the instructions provided to bring up the Main Menu.
 - **World Wide Web Home Page:** The STSC Web Home Page utilizes the most recent methods for locating information on the Internet. Initially, our Web site was experimental in nature, but is now maturing into a productive and useful site. The number of software engineering options are expanding, as are the hot links and menus featuring other Web Servers.
 - **Lynx Browser:** The BBS Main Menu now features a Lynx Browser option. It is tailored for users desiring to access and explore the vast information resources of the World Wide Web, but who lack full Web and graphics capability. It was developed by the University of Kansas as a text only Web browser. Lynx brings the Web to an individual's personal computer in ASCII text only, but is a tested and proven, simple to use Web navigational tool. To access the STSC's Lynx Browser, Telnet to or dial into the BBS as explained above. Select Option [14] - Lynx Browser to WWW Server, on then STSC On-Line Services Main Menu.

1.2.3 TECHNOLOGY INSERTION PROJECTS

STSC technology insertion projects are customer-oriented projects that evaluate, select, and pilot the use of new processes, methods, and technologies for a specific customer. These projects can include process definition, process improvement, methodology insertion, tool insertion, and development of a technology road map. Referring to the Adoption Curve (Figure 1-1), an insertion project helps cement understanding of a process or technology, tailors an evaluation of the process or technology for the customer, and pilots the use of that process or technology with appropriate levels of training. Customers move closer to adoption of the process or technology through hands-on experience. It is important to try out technology improvements in a pilot project to confirm that the technology is appropriate for the organization and that the organization is ready and able to adopt the new technology.

1.2.4 STSC ASSOCIATES

Fowler and Przybylinski [Fowler 88] propose that transitioning new technologies from a developer to a consumer requires an advocate to push the technology and a receptor to pull the technology into an organization. This concept is illustrated in Figure 1-2.

Effective change comes from within the organization. The STSC Associates objective is to develop technology receptors within individual Air Force SDSAs. These receptors, STSC Associates, are trained in the use of the STSC's information, products, and services to enhance their organization's ability to incorporate advanced practices, processes, and technologies.

Referring to the adoption curve in Figure 1-1, STSC Associates complete the trek to institutionalization. Associates who come from within the organization should be politically astute and aware of internal organizational requirements. They have the highest probability of influencing the adoption and daily use of effective business practices, processes, and technologies.

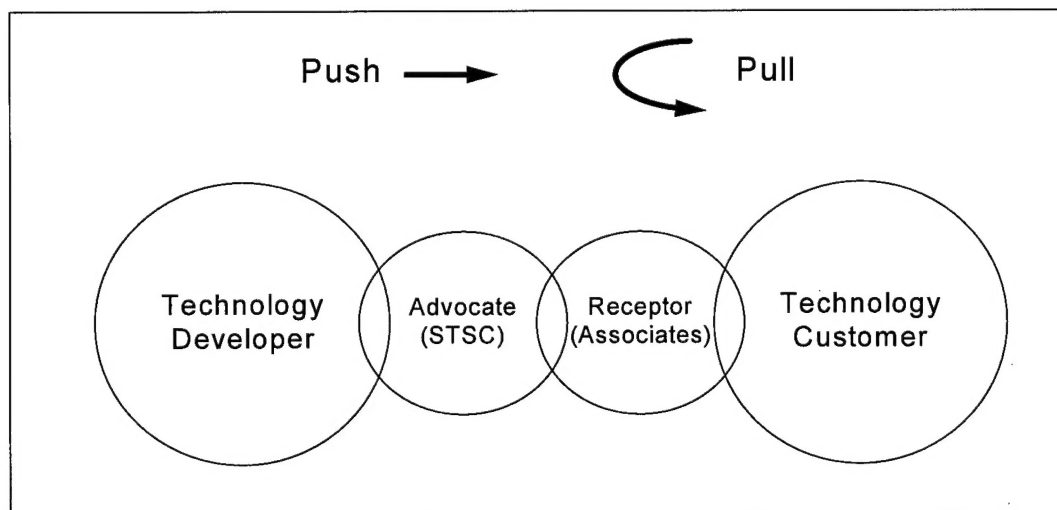


Figure 1-2. Transitioning Technology.

1.3 EMBEDDED COMPUTER RESOURCES SUPPORT IMPROVEMENT PROGRAM (ESIP)

The STSC operates from the Ogden Air Logistics Center at Hill Air Force Base, Utah, under the direction and guidance of the ESIP Program Office. An Air Force program, the ESIP has the goals to reduce the software backlog and increasing software quality and productivity. Its mission is to provide an infrastructure to assist in the transitioning of technology to support all categories of embedded computer systems throughout the acquisition cycle and improve the readiness of Air Force weapon systems. ESIP is divided into four tasks: Readiness, the Software Technology Support center (STSC), Extendible Integration Support Environment (EISE), and Advanced R&D. ESIP is directed by an Air Force program management directive (PMD3118) and is led by Col. Charles Fuller. An ESIP working group has been established as a forum to share lessons learned and establish requirements for ESIP funded technology transition projects. Working group members are from the major commands, ESIP task managers, and the ESIP program office staff.